第五章 数据降维

5.1 降维的基本概念

5.1.1 降维的定义及原理

学习降维，首先要理解数据降维的基本概念。数据降维一般用于减少数据矩阵的特征数量，用线性或者非线性的映射把样本由高维映射到低维，目的是减少冗余和噪声特征和提高模型泛化能力和计算效率。
其中特征值和特征向量在降维算法中具有重要意义，如在奇异值分解、主成分分析等算法里，特征值和特征向量的计算是实现降维的关键步骤。它们能够反映矩阵在不同方向上的 “作用” 程度，即数据在对应特征向量方向上的变化情况。特征值越大，表明矩阵在该特征向量方向上的变化幅度越大，数据在这个方向上的方差也就越大。
降维具有如下一些优点：

- 使得数据集更易使用。
- 降低算法的计算开销。
- 去除噪声。
- 使得结果容易理解。

高维数据是指特征数量p大于观测数量N的数据集，通常写为p >> N。例如，具有p = 6 个特征且只有N = 3 个观测值的数据集将被视为高维数据，因为特征数量大于观测值数量。人们常犯的一个错误是假设“高维数据”仅仅意味着具有许多特征的数据集。然而，这是不正确的。一个数据集可能包含 10,000 个特征，但如果包含 100,000 个观测值，则它不是高维的。
为什么高维数据是一个问题？当数据集中的特征数量超过观察数量时，我们永远不会得到确定性的答案。换句话说，不可能找到一个可以描述预测变量 和响应变量之间关系的模型，因为我们没有足够的观测数据来训练模型。
高维数据集在医疗保健、金融、基因组学等领域中较为常见。
在大数据背景下，高维数据的挑战：

- 样本稀疏性：高维空间中数据分布稀疏，导致欧式距离失效。
- 计算复杂度：特征数量增加导致计算和存储成本上升。
- 噪声与冗余：高维数据易受无关特征干扰，模型易过拟合。
- 维数灾难(Curse of Dimensionality)：在涉及到向量的计算的问题中，随着维数的增加，计算量呈指数倍增长的一种现象。维数灾难涉及数字分析、抽样、组合、机器学习、数据挖掘和数据库等诸多领域。维数灾难由 Richard E. Bellman 提出，是高维数据处理中面临的重大挑战。随着（数学）空间维数增加，估计多变量函数所需的采样点数会呈指数增长。以气象数据为例，早期可能仅用温度、湿度两个特征描述天气，随着气象学发展，增加了气压、风力、降雨量、辐射强度等众多特征，数据维数大幅增加。此时，计算数据集的相似性、估计模型参数等操作变得极为困难和耗时。同时，计算复杂度与存储

需求也显著增加，模型预测准确度下降，这些问题严重阻碍了数据处理工作的进行，在这些问题中“维数灾难” 是比较常遇见的问题。

那么在此情况下，我们应该如何处理高维数据呢？这时候就用到了数据降维技术，降维技术可以通过把高维数据映射到更低维的空间，保留数据主要结构和信息，提高模型泛化能力。

5.1.2 降维算法分类

数据降维过程中主要用到两种方法：

- 线性降维方法：特点是假设数据线性可分，通过正交变换实现降维。当数据集的各个属性（特征）独立无关，或者非线性数据可以在一定程度上用线性结构近似表达时，适合采用线性降维方法。常见的线性降维算法包括奇异值分解（SVD）、主成分分析（PCA）、因子分析（FA）、线性判别分析（LDA）等。这些算法通过线性变换将高维数据映射到低维空间，在降低维度的同时，尽可能保留数据的主要信息。以主成分分析为例，它通过正交变换将原有数据的特征变量线性组合为一组线性无关的综合指标，实现用少量综合指标来表示原有数据的大部分信息。
- 非线性降维方法：特点是处理非线性结构，基于流形学习或概率分布保持局部/全局关系。当数据为高度非线性或数据属性存在较强的相关性时，线性降维方法往往难以取得理想效果，此时需要运用非线性降维方法。这类方法通过引入核函数、流形学习或神经网络等技术来实现降维。常见的非线性降维算法有多维尺度变换（MDS）、等距特征映射（IsoMap）、t-SNE、UMAP 等。例如，等距特征映射基于流形学习的思想，利用测地距离来保持降维前后数据点间的相似性，能够还原出数据集的真实低维流形结构，适用于处理具有复杂非线性结构的数据。

在大数据时代，数据规模不断扩大，高维数据带来的问题日益突出。降维技术成为解决这些问题的有效手段，具有多方面的重要意义。在计算负担方面，降维能够减少数据处理所需的计算资源和时间。以推荐系统为例，用户 - 物品评分矩阵维度很高，通过降维可以降低计算复杂度，提高推荐算法的效率。在模型性能上，降维能够去除冗余和噪声特征，使模型更加聚焦于关键信息，从而提高模型的准确性和泛化能力。在发现隐藏模式方面，降维可以帮助挖掘数据中的潜在规律和模式。例如在基因数据分析中，降维技术有助于发现基因之间的潜在关系，为生物研究提供有价值的信息。

5.2 奇异值分解

5.2.1 奇异值分解基本原理

（一）奇异值分解算法

奇异值分解（SVD）是一种强大的矩阵分解技术，能将高维矩阵表示为三个低维矩阵的乘积。许多数学对象可以通过将它们分解成多个组成部分或者找到它们的一些属性来更好地理解，这些属性是通用的,而不是由我们选择它们的方式所产生的。例如，我们可以通过分解质因数来发现整数的一些内在性质。尽管我们可以用十进制或者二

进制等不同方式表示整数12.但是12=2×3×3总是对的。从这个表示中我们可以获得一些有用的信息，比如12不能被5整除，或者12的倍数能够被3整除。对应地，我们也可以通过分解矩阵来发现矩阵表示成数组元素时不明显的函数性质。特征分解是使用最广的矩阵分解之一，在这个分解中，我们将矩阵分解成一组特征向量和特征值。这样可以帮助我们分析矩阵的特定性质，就像质因数分解有助于我们理解整数。

此外，我们还可以对奇异值分解做出几何解释。矩阵的奇异值分解也可以看做是：一个坐标系的旋转或反射变换、一个坐标轴的缩放变换、另一个坐标系的旋转或反射变换。奇异值定理保证这种分解一定存在，这就是奇异值分解的几何解释。

奇异值分解是将矩阵分解为奇异向量和奇异值。通过奇异值分解，我们会得到一些与特征分解相同类型的信息。然而，奇异值分解有更广泛的应用。每一个实数矩阵都有一个奇异值分解，但不一定都有特征分解（例如非方阵矩阵）。

在使用线性代数的地方，基本上都要使用 SVD。 SVD 不仅仅应用在 PCA 、图像压缩、数字水印、 推荐系统和文章分类、 LSA (隐性语义分析)、特征压缩(或数据降维)中，在信号分解、信号重构、信号降噪、数据融合、同标识别、目标跟踪、故障检测和神经网络等方面也有很好的应用， 是很多机器学习算法的基石。在实际应用中，如在图像处理里，图像可以表示为一个矩阵，通过 SVD 分解，能将图像矩阵分解为三个矩阵。这三个矩阵分别从不同方面描述了图像的特征，为后续的图像压缩、特征提取等操作提供了便利。

（二）奇异值分解原理推导

- 求解左、右奇异矩阵：由于数据量 N 远大于数据维数 p，对于数据矩阵 A，无法直接进行特征值分解。SVD 算法通过求解矩阵的特征值与特征向量来间接实现分解。通过求解矩阵的特征值及对应的特征向量矩阵，得到左、右奇异矩阵U、Σ和奇异值矩阵 Σ。
- 奇异值分解结果：通过上述步骤求解出矩阵 U、Σ、V 后，将分解结果写成向量形式，得到 A 。奇异值分解的过程就像是将一个复杂的矩阵 “拆解” 成几个相对简单的矩阵，每个矩阵都承载着原矩阵不同方面的信息，为后续的数据处理和分析提供了基础。

尝试练习使用numpy进行矩阵的奇异值分解代码：

```
import numpy as np
# 创建矩阵A
A = np.array([[1, 0, 0, 0],
              [0, 0, 0, 4],
              [0, 3, 0, 0],
              [0, 0, 0, 0],
              [2, 0, 0, 0]])

# 进行奇异值分解
U, s, V = np.linalg.svd(A)
# 打印结果
```

```
print("U:\n", U)
print("s:", s)
print("V:\n", V)
```

输出结果：

```
U:
 [[ 0.          0.         -0.4472136   0.         -0.89442719]
 [-1.          0.          0.          0.          0.        ]
 [ 0.         -1.          0.          0.          0.        ]
 [ 0.          0.          0.          1.          0.        ]
 [ 0.          0.         -0.89442719  0.          0.4472136 ]]
s: [ 4.          3.          2.23606798 -0.        ]
V:
 [[-0. -0. -0. -1.]
 [-0. -1. -0. -0.]
 [-1. -0. -0. -0.]
 [-0. -0. -1. -0.]]
```

截断奇异值分解是比原始矩阵低秩的奇异值分解。在矩阵的奇异值分解中，只取最大的k个奇异值（k小于r,r为矩阵的秩）对应的部分，就得到矩阵的截断奇异值分解。实际应用中提到矩阵的奇异值分解时，通常指截断奇异值分解。

在此，我们进行截断奇异值分解的numpy代码实现练习：

```
import numpy as np
# 创建矩阵A
A = np.array([[1, 0, 0, 0],
              [0, 0, 0, 4],
              [0, 3, 0, 0],
              [0, 0, 0, 0],
              [2, 0, 0, 0]])

# 进行截断奇异值分解
k = 2  # 指定要保留的前 k 个奇异值
U, s, V = np.linalg.svd(A, full_matrices=False)
```

```
# 仅保留前 k 个奇异值及相应的奇异向量
U_k = U[:, :k]
s_k = s[:k]
V_k = V[:k, :]

# 构建紧奇异值分解的近似矩阵
A_k = U_k.dot(np.diag(s_k)).dot(V_k)

# 打印结果
print("U_k:\n", U_k)
print("s_k:", s_k)
print("V_k:\n", V_k)
print("A_k:\n", A_k)
```

输出结果：

```
U_k:
 [[ 0.  0.]
 [-1.  0.]
 [ 0. -1.]
 [ 0.  0.]
 [ 0.  0.]]
s_k: [4. 3.]
V_k:
 [[-0. -0. -0. -1.]
 [-0. -1. -0. -0.]]
A_k:
 [[0. 0. 0. 0.]
 [0. 0. 0. 4.]
 [0. 3. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

5.2.2 奇异值分解算法评价

（一）优点

- 降低数据维数：SVD 能够有效降低数据的维数，将高维数据压缩到低维空间，减少数据处理的复杂性。在推荐系统中，用户 - 物品评分矩阵通常是高维稀疏矩阵，使用 SVD 可以将其降维，减少存储和计算成本。
- 节约计算时间：通过分解为低维矩阵的乘积，在模型训练时可以大大节约计算时间，提高算法效率。在大规模数据处理中，这一优势尤为明显，能够使算法更快地收敛，提高系统的响应速度。
- 应用广泛：SVD 常用于推荐系统、图像处理、自然语言处理等多个领域。在图像处理中，可用于图像压缩和特征提取；在自然语言处理中，可用于词嵌入和语义分析，展现了其强大的通用性和实用性。

（二）缺点

SVD 算法分解后得到的结果（奇异值和奇异矩阵），无法直接对应于原始数据的特征，导致降维后的数据可读性与可解释性较低。在实际应用中，这可能会给数据分析和理解带来一定困难，需要结合其他方法进行进一步的处理和分析。

我们可以了解一下关于奇异值分解在推荐系统中的应用，其主要基于模型的协同过滤。SVD 通过对用户项目评分矩阵进行分解，捕获用户和项目之间的潜在关系，从而预测用户对未评分项目的潜在评分，进行个性化推荐。

以下是应用 SVD 的具体步骤：

- 构建评分矩阵：首先构建一个用户-项目评分矩阵，其中每行表示一个用户，每列表示一个项目，矩阵中的元素表示用户对项目的评分。通常，这个矩阵是稀疏的，因为用户只对部分项目进行了评分。
- 计算奇异值分解：对评分矩阵进行奇异值分解，得到 U, Σ 和 V^T。U 和 V 分别表示左奇异向量和右奇异向量矩阵，而 Σ 是一个对角矩阵，其对角线上的元素表示奇异值。
- 选择潜在因子数量：选择一个减小维度的值 k，用于表示潜在因子的数量。选择较小的 k 值可以实现压缩评分矩阵，但过小的 k 可能损失较多信息。选择合适的 k 值需要在压缩程度和损失信息之间取得平衡。
- 重构低秩评分矩阵：通过截取 U、Σ 和 V^T 中前 k 个奇异值及相应的向量，重新构建近似评分矩阵。这个近似矩阵将捕获用户和项目之间的主要潜在关系，但使用较少的存储空间。
  预测未评分项目：使用重构的评分矩阵来预测用户对尚未评分项目的评分。通过比较重构矩阵中用户行和项目列的值，可以估计用户对每个项目的潜在评分。
- 计算推荐：根据预测的评分，为每个用户推荐具有最高潜在评分的项目。

SVD 能有效找到原始评分矩阵中的潜在关系，从而实现高效的推荐。不过，在推荐系统实际应用中，奇异值分解可能需要面临大规模且稀疏的矩阵。

5.2.3 实践案例：基于奇异值分解的图像压缩

（一）模型构建

在 Python 中，SVD 算法可以通过使用 NumPy 库中 linalg 模块的 svd 函数实现。在进行图像压缩时，对一张 RGB 图像进行奇异值分解，构建 restore 函数用于图像重构，并展示重建图像随奇异值数量变化的效果，如下。

代码：

```
import numpy as np
import os
from PIL import Image
import matplotlib.pyplot as plt
#  1.定义恢复函数，由分解后的矩阵恢复到原矩阵
def restore(u, s, v, K):
    n, p = len(u), len(v[0])
    a = np.zeros((n, p))
    for k in range(K):
        uk = u[:, k].reshape(n, 1)
        vk = v[k].reshape(1, p)
        a += s[k] * np.dot(uk, vk)
    a = a.clip(0, 255)
    return np.rint(a).astype('uint8')
A = np.array(Image.open("D:/3.研究生课程/大数据分析/降维/data/svd.jpg", 'r'))
#  2.对RGB图像进行奇异值分解
u_r, s_r, v_r = np.linalg.svd(A[:, :, 0])
u_g, s_g, v_g = np.linalg.svd(A[:, :, 1])
u_b, s_b, v_b = np.linalg.svd(A[:, :, 2])
#  3.保存奇异值为10，20，30，40，50的图像，并输出特征数
selected_svd_values = [10, 20, 30, 40, 50]
for k in selected_svd_values:
    R = restore(u_r, s_r, v_r, k)
    G = restore(u_g, s_g, v_g, k)
    B = restore(u_b, s_b, v_b, k)
    I = np.stack((R, G, B), axis=2)
    Image.fromarray(I).save(f'svd_{k}.jpg')
#  4.可视化重构图像
fig, axes = plt.subplots(1, 5, figsize=(20, 8))
for i, k in enumerate(selected_svd_values):
    img = Image.open(f'svd_{k}.jpg')
    axes[i].imshow(img)
    axes[i].set_title(f'K = {k}')
plt.show()
```

（二）参数调整

参数 “a” 代表输入一个 N×p 阶数据矩阵；“full_matrices” 表示是否返回完整的奇异矩阵，取值为 “True” 时，返回的左奇异矩阵为 N 阶方阵，右奇异矩阵为 p 阶方阵；取值为 “False” 时，返回的左、右奇异矩阵分别为 N×k、k×p 阶矩阵，其中 k = min (N, p) 。通过调整这些参数，可以根据实际需求获取不同形式的奇异矩阵，以满足不同的应用场景。

5.3 主成分分析

5.3.1 主成分分析基本原理

（一）基本思想

主成分分析是一种广泛应用的无监督降维算法，它是最常用的降维算法之一，可以很好的解决因变量太多而复杂性，计算量增大的弊端。它采用线性变换将一组统计变量（指标）转换为新的一组综合变量，即主成分。这些主成分之间互不相关，且着重解释各变量的总方差，旨在用少量的主成分尽可能代表原来变量的信息。本质上讲，PCA就是将高维的数据通过线性变换投影的方式映射到低维空间上去，并且保证在投影的维度上，原数据的信息量最大（损失最小）。例如在分析学生综合成绩时，学生的成绩可能涉及多门课程，这些课程成绩可看作多个变量。PCA 能将这些变量线性组合成少数几个主成分，通过分析主成分来综合评估学生的能力，简化数据结构的同时保留关键信息。

我们如何得到这些包含最大差异性的主成分方向呢？

事实上，通过计算数据矩阵的协方差矩阵，然后得到协方差矩阵的特征值特征向量，选择特征值最大(即方差最大)的k个特征所对应的特征向量组成的矩阵。这样就可以将数据矩阵转换到新的空间当中，实现数据特征的降维。

由于得到协方差矩阵的特征值特征向量有两种方法：特征值分解协方差矩阵、奇异值分解协方差矩阵，所以PCA算法有两种实现方法：基于特征值分解协方差矩阵实现PCA算法、基于SVD分解协方差矩阵实现PCA算法。

（二）理论推导

对主成分分析进行理论推导，我们知道PCA有两种通俗易懂的解释：(1)最大方差理论；(2)最小化降维造成的损失。最大方差解释是PCA中的一个重要概念，它是指在PCA过程中选择方差最大的投影方向。最大方差理论认为，方差是衡量数据分散程度的一个指标，方差越大，意味着该方向上的数据分散程度越高，包含的信息量越大。因此，在PCA中应选择方差最大的投影方向，以保留更多的原始数据信息。

- 构建主成分分析模型：在实际应用中，若数据特征量纲不同或相同量纲下数据值差异过大，直接进行主成分分析会导致结果偏差。因此，一般先对原始数据进行标准化处理，确保各个变量在相同尺度上比较分析，使求解的主成分具有实际意义。
- 求解主成分方差最大化问题：为使主成分能充分反映原变量信息，需满足主成分的方差尽可能大且相互独立。
- 主成分的性质：主成分的方差依次递减；主成分之间相互独立；每个主成分的系数平方和为 1。

（三）主成分分析的步骤

（1）将原始数据标准化，求解标准化后的数据矩阵的协方差矩阵的特征值，对其进行降序排列；

（2）建立变量的相关系数阵；

（3）求特征根为λ1≥…≥λp≥0，相应的特征向量为T1,T2,…,TP，从而得到各特征值对应的单位化特征向量矩阵；

（4）由累积方差贡献率确定主成分的个数（m）；

（5）得到降维后的主成分。

下面，我们可以使用Python练习关于主成分分析的代码：

```
##Python实现PCA
import numpy as np
def pca(X,k):#k is the components you want
  #mean of each feature
  n_samples, n_features = X.shape
  mean=np.array([np.mean(X[:,i]) for i in range(n_features)])
  #normalization
  norm_X=X-mean
  #scatter matrix
  scatter_matrix=np.dot(np.transpose(norm_X),norm_X)
  #Calculate the eigenvectors and eigenvalues
  eig_val, eig_vec = np.linalg.eig(scatter_matrix)
  eig_pairs = [(np.abs(eig_val[i]), eig_vec[:,i]) for i in range(n_features)]
  # sort eig_vec based on eig_val from highest to lowest
  eig_pairs.sort(reverse=True)
  # select the top k eig_vec
  feature=np.array([ele[1] for ele in eig_pairs[:k]])
  #get new data
  data=np.dot(norm_X,np.transpose(feature))
  return data

X = np.array([[-1, 1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])

print(pca(X,1))
```

上面代码可以对数据X进行特征的降维。

5.3.2 主成分分析算法评价

（一）优点

- 有效去除噪声：原始数据的低方差主成分中通常存在噪声，PCA 算法保留高方差的主成分并过滤掉较小方差的主成分，能够有效地去除噪声并保留数据的主要特征。在图像处理中，可通过 PCA 去除图像中的噪声，提高图像质量。
- 广泛的应用领域：PCA 算法常用于特征工程、图像处理、数据压缩和模式识别等领域。在特征工程中，可减少特征数量，提高模型训练效率；在数据压缩方面，可对高维数据进行降维压缩，减少存储空间。

（二）缺点

PCA 算法适用于变量间有较强相关性的数据。如果原始数据相关性较弱时，方差小的成分可能含有影响样本差异的重要信息，降维丢弃这些成分可能对后续数据处理产生影响，导致主成分分析算法不能起到很好的降维效果 。在分析某些特殊数据时，若变量间相关性不明显，使用 PCA 降维可能会丢失关键信息，影响数据分析结果。

5.3.3 实践案例：基于主成分分析的红酒数据集分析

（一）模型构建

在 Python 中，PCA 算法可通过 Scikit-learn 中 decomposition 模块的 PCA 函数实现。操作步骤包括创建 PCA 对象并指定降维的维数，然后拟合数据并对数据进行转换，最终得到降维后的结果，如下。

代码：

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib
# 1.读取数据
red_wine_df = pd.read_csv('D:/3.研究生课程/大数据分析/降维/data/winequality-red.csv', index_col
=0)
red_wine_df.isnull().sum()
# 2.对数据进行标准化处理
scaler = StandardScaler()
red_wine_scaled = pd.DataFrame(data=scaler.fit_transform(red_wine_df), columns=red_wine_df.colu
mns)
# 3.聚类
kmeans = KMeans(n_clusters=3)
```

```
clusters = kmeans.fit_predict(red_wine_scaled)
# 4.降维至二维
pca = PCA(n_components=2)
pca_2D = pca.fit_transform(red_wine_scaled)
# 将PCA结果和聚类结果合并到一个DataFrame
pca2D_df = pd.DataFrame(data=pca_2D, columns=['PC1', 'PC2'])
pca2D_df['cluster'] = clusters
# 5.可视化降维结果
plt.rcParams['font.sans-serif']=['Times New Roman']
plt.rcParams.update({'font.size': 20})
plt.figure(figsize=(6, 5))
sns.scatterplot(x='PC1', y='PC2', hue='cluster', data=pca2D_df, s=20)
plt.tick_params(axis='both', which='major')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend(frameon=False)
plt.tight_layout()
plt.show()
```

（二）参数调整

函数 PCA 中的参数 “n_components” 表示 PCA 算法中所要保留的主成分个数；参数 “whiten” 对数据集每个特征进行缩放，使它们具有相同的方差，默认取值为 “False”；参数 “svd_solver” 代表奇异值分解方法，默认取值 “auto” 表示根据输入数据的大小和特征数量来自动选择解决方法，取值 “full” 表示使用传统的基于全矩阵运算的奇异值分解方法，取值 “randomized” 表示使用随机化的 SVD 方法进行奇异值分解 。在实际应用中，可根据数据特点和需求调整这些参数，以获得更优的降维效果。

5.4 因子分析

5.4.1 因子分析基本原理

（一）基本思想及涉及概念

因子分析可以看成是主成分分析的推广与发展。在PCA中，我们是将特征值从大到小排列，据此来选择对应的特征向量；而在此处（FA），特征值（根）同时也代表着相应公因子的方差贡献大小。因子分析法通过研究变量间的相关系数矩阵，把这些变量间错综复杂的关系归结成少数几个综合因子，由于归结出的因子个数少于原始变量的个数，但是它们又包含原始变量的信息，所以，这一分析过程也称为降维。由于因子往往比主成分更易得到解释，故因子分析比主成分分析更容易成功，从而有更广泛的应用。

更加专业地说，因子分析（Factor Analysis，FA）是一种线性降维方法，旨在用较少几个因子表示原始数据的大

部分信息。其核心是把每个研究变量分解为几个影响因素变量，将原始变量分解为两部分：一部分是由所有变量共同具有的少数几个公共因子组成，另一部分是每个变量独自具有的特殊因子。例如在分析学生体育测试成绩时，100 米、400 米、标枪、铅球和跳高这五个项目的成绩可视为原始变量。因子分析认为这些项目主要考察学生的爆发力、耐力、力量和灵敏性等潜在能力，这些潜在能力就是公共因子，而每个项目可能还受一些特殊因素影响，如 100 米项目的起跑反应速度对该项目成绩的特殊影响，这就是特殊因子。通过这种分解，因子分析能够挖掘数据背后的潜在结构，简化数据的复杂性。

接下来，我们学习一下一般因子分析模型：设有 n个样品，每个样品观测 p个指标，这 p个指标之间有较强的相关性（要求 p个指标相关性较强的理由是很明确的，只有相关性较强才能从原始变量中提取出“公共”因子）为了消除由于观测量纲的差异及数量级不同所造成的影响，将样本观测数据进行标准化处理，使标准化后的变量均值为0，方差为1。为方便，把原始变量及标准化后的变量向量均用X 表示，用F1，F2，...，Fm（m ＜p）表示标准化的公共因子。

而关于因子载荷矩阵，我们了解到X是原始数据的矩阵，f是公共因子，也就是之后想要提取出来的公共因子，而特殊因子就是前面的公共因子没有包涵的部分，实际分析的时候可以不管。那么，什么是因子载荷矩阵？百度百科：因子载荷 aij 的统计意义就是第i个变量与第 j 个公共因子的相关系数即表示 Xi 依赖 Fj 的份量（比重）。统计学术语称作权，心理学家将它叫做载荷，即表示第 i 个变量在第 j 个公共因子上的负荷，它反映了第 i 个变量在第 j 个公共因子上的相对重要性。同时，aij也是我们所构建的因子分析模型X = Af+ε中的矩阵A的元素，也就是模型中各个因子F的系数。

基于此我们得到：

- 第一行的元素分别代表着第一个变量和第一个公因子的相关系数、第一个变量和第二个公因子的相关系数...第一行元素的平方和hi^2代表着第一个变量和所有公因子的R方之和，换个角度也可以理解为所有公共因子对第一个变量的解释程度。（称为变量共同度，也可以称为公因子提取度）
- 同理，第一列元素分别代表第一个变量和第一个公因子的相关系数、第二个变量和第一个公因子的相关系数...第一列元素的平方和gj^2代表着第一个公因子对所有变量的影响，因此“也是对比公共因子重要性的一个标准”。（称为公因子的方差贡献）变量共同度和公因子方差贡献这两个概念十分重要！后面会一直用到！
- 综上，A的元素的平方和代表所有公共因子对总方差的累计贡献。

那么，因子载荷矩阵A是如何得到的呢？最常用的方法是基于样本相关系数矩阵R的主成分解得到的。至于相关系数矩阵，它就是在主成分分析中讲过的协方差矩阵的升级版（协方差去除量纲后得到相关系数）。

（二）因子分析的具体步骤

- 考察数据是否适合进行因子分析，主要是通过KMO检验和Bartlett球形检验。
- 计算相关系数矩阵，并由此得到因子载荷矩阵。
- 对载荷矩阵进行因子旋转。

  因子分析的目的不仅仅是要找出公共因子以及对变量进行分组，更重要的是要知道每个公共因子的意义，以便进行进一步的分析。如果每个公共因子的含义不清，则不便于进行实际背景的解释。

  初始因子（也就是我们从上一步算出来的因子载荷矩阵）的综合性太强，难以找出因子的实际意义。由

于因子载荷阵是不唯一的，所以可以对因子载荷阵进行旋转，使因子载荷阵的结构简化，从而使因子的实际意义更容易被解释。

- 计算因子得分

  通过上面的因子旋转后，我们得到了最终的简化因子载荷阵。但是大多时候我们需要使用这些因子做其他的研究。比如把得到的因子作为自变量来做回归分析，对样本进行分类或评价，这就需要我们对公共因子进行测度，即给出公共因子的值，而不是仅仅只有一个矩阵。

  计算因子得分的方法主要有两种方法：巴特莱特因子得分（加权最小二乘法）和回归法。巴特莱特因子得分比较好理解，就像是我们平常做OLS回归中，先待定回归系数，然后用大量的数据估计出回归系数。而此处就是把得分Fj当做回归系数来估计。综合因子得分的计算就等于(各个因子得分乘以其方差贡献率)/各个因子的累计贡献率之和。

接下来详细学习一下适用性检验和因子旋转。

（三）适用性检验

因子分析的目的是在互为相关的许多变量中寻找能反映它们之间内在联系以及起主导作用的、数目较少的因子，通过对这些因子的研究,既无损于原来多个变量的信息，又便于对它们进行分类和解释。因此，要想使用因子分析方法，其前提是:原始数据中多个变量之间应有较强的线性相关关系。如果原始变量之间的线性相关程度太小，它们之间就不存在具有说服力的公因子，这时进行因子分析就没有实际意义；如果各个变量之间相互独立，这时多变量的协方差矩阵是对角阵（更确切地说是单位阵，因为若各个变量之间相互独立，则协方差矩阵除了对角线之外的元素都为0，而对角线上的元素代表自己和自己的协方差，为1。因此各个变量之间相互独立的协方差矩阵为单位阵），显然这样的数据是不适合用因子分析进行分析。

- 先简单介绍Bartlett球形检验的原理：
  1. 巴特利特球形检验法是以相关系数矩阵为基础的.它的零假设相关系数矩阵是一个单位阵,即相关系数矩阵对角线的所有元素均为1,所有非对角线上的元素均为零.巴特利特球形检验法的统计量是根据相关系数矩阵的行列式得到的.如果该值较大,且其对应的相伴概率值（即p值）小于指定的显著水平时,拒绝零假设,表明相关系数矩阵不是单位阵,原有变量之间存在相关性,适合进行主成分分析；反之,零假设成立,原有变量之间不存在相关性,数据不适合进行主成分分析。
  2. 球形检验主要是用于检验数据的分布，以及各个变量间的独立情况。按照理想情况，如果我们有一个变量，那么所有的数据都在一条线上。如果有两个完全独立的变量，则所有的数据在两条垂直的线上。如果有三条完全独立的变量，则所有的数据在三条相互垂直的线上。如果有n个变量，那所有的数据就会在n条相互垂直的线上，在每个变量取值范围大致相等的情况下（常见于各种调查问卷的题目），所有的数据分布就像在一个球形体里面，大抵就是那个样子。如果不对数据分布进行球形检验，在做因素分析的时候就会违背因素分析的假设——各个变量在一定程度上相互独立。
- KMO检验的原理：

  如果原始数据中确实存在公共因子，则各变量之间的偏相关系数应该很小，这时KMO的值更容易向1趋近，因此，原数据适用于因子分析。

为什么存在公共因子各个变量之间的偏相关系数就会很小的情况？

在KMO检验中，我们将变量之间的相关性分为两部分：一部分是由其他变量解释的共同变异性，另一部分是独特的变异性，即不能由其他变量解释的部分。当存在公共因子时，变量间的相关性主要由公共因子驱动。在控制其他变量（作为公共因子的代理）后，偏相关系数反映的是独特因子间的剩余相关性。由于独特因子之间相互独立（或弱相关），偏相关系数自然较小。

（四）因子旋转原理解释及具体步骤

因子旋转的正交旋转（除了正交还有斜交，但用的较少，此处不予讨论）中，主要有三种方法，分别为：

1. 方差最大旋转法（Varimax）；
2. 四次方最大旋转法（Quartimax）；
3. 等量最大旋转法（Equamax）；

此处仅了解方差最大法。方差最大旋转法从简化因子载荷阵的每一列出发，使和每个因子有关的载荷平方的方差最大，当只有少数几个变量在某个因子上有较高的载荷时候，对因子的解释最简单。这段话如何理解呢？首先简化因子载荷阵就是我们将载荷矩阵经过因子旋转后得到的新的载荷矩阵，由于这个矩阵是较为简单的，故以此命名。当然，我们现在还不知道如何通过因子旋转得到这个简化因子载荷阵，只是此处采用的是类似待定系数的思想，我们先假设有这么一个东西存在。

其次，从一个因子载荷阵（不管是不是简化的）的每一列来看，我们前面有讲过列元素的平方和gj^2称为公因子的方差贡献。然后接下来的这整段话“使和每个因子有关的载荷平方的方差最大，当只有少数几个变量在某个因子上有较高的载荷时候，对因子的解释最简单”的意思是：假设我们有四个变量abcd，两个公因子αβ，比如理想的两极分化的情况是α和ab完全相关，和cd完全无关；β和ab完全无关，和cd完全相关。这样就满足了简化情况的要求，而不是abcd四个变量和αβ都有关系，从而让我们很难去解释α和β这两个因子的含义。方差最大的直观意义是希望通过因子旋转后，使每个因子上的载荷尽量拉开距离，一部分的载荷趋于±1，另一部分趋于0。那方差和上述所说的有什么关系呢？为实现使各个因子上的载荷两极分化，使得因子载荷之间差异极大化，须让描述差异性的统计指标—方差极大化。

正交变换后的公因子共同度不变，因子方差贡献发生变化。因为公因子共同度代表的是全部公因子对某一个变量的影响，公因子共同度不变则代表着全部公因子对某一个变量的影响之和不变；而因子方差贡献代表着某个公因子对所有变量的影响，也就是某个因子对总体解释贡献度的大小。由于做了因子旋转，旋转后的因子和之前的因子已经不一样了，所以每个因子的贡献大小和之前不一样，也是很正常的事情。

最后总结一下因子旋转中方差最大化旋转的步骤和原理。

- 步骤：

  （1） 设已求出的因子载荷矩阵A

  （2） 现选取正交变换矩阵 Γ 进行因子旋转

  （3） 求θ。这里θ是坐标平面上因子轴按顺时针方向旋转的角度，只要求出θ，也就求出了Γ。

  （4） 重复旋转过程。选择不同的因子，重复步骤（2）和（3），不断进行旋转操作。每一次旋转都会改变因子载荷矩阵，使得目标函数逐渐增大。

（5）收敛判断。持续进行旋转操作，直到目标函数的变化小于某个预设的阈值，即认为达到了收敛条件。此时停止旋转，得到的因子载荷矩阵就是经过方差最大化旋转后的最终结果。

（6）结果解释。根据旋转后的因子载荷矩阵，分析每个因子所对应的高载荷变量，从而确定每个因子的实际含义。通常将在某个因子上载荷较高（如绝对值大于 0.7）的变量归为该因子所代表的类别，这样就可以更清晰地解释数据背后的潜在结构。

- 原理：

  方差最大化旋转的核心原理是通过对因子载荷矩阵进行正交变换，使得每个因子上变量载荷平方的方差达到最大。其目的在于让每个因子所代表的变量更加清晰和集中，从而提高因子的可解释性。在初始的因子分析中，因子载荷矩阵可能呈现出多个变量在多个因子上都有一定载荷的情况，这使得因子的含义难以明确界定。方差最大化旋转通过调整因子轴的方向，使得部分变量在某个因子上的载荷趋近于 1，而其他变量在该因子上的载荷趋近于 0，从而将变量更好地归类到不同的因子中。

（五）阐述因子分析与主成分分析的基本思想、联系和区别

因子分析的基本思想：是一种数据简化技术，通过研究众多变量之间的内部依赖关系，探求观测数据的基本结构，并用少数几个假想变量（因子）来表示原始数据。

主成分分析的基本思想：当第一个线性组合不能提取更多的信息时，再考虑用第二个线性组合继续这个快速提取的过程，……，直到所提取的信息与原指标相差不多时为止。

- 联系：

  无论是R型或Q型因子分析，都用公共因子F代替X，一般要求m小于p，m小于n，因此，因子分析与主成分分析一样，也是一种降低变量维数的方法。因子分析的求解过程同主成分分析类似，也是从一个协方差阵出发的。综上，主成分分析和因子分析都是常用的降维技术，在数据处理过程中都致力于减少变量的数量，以简化数据分析。它们都基于线性变换的思想，通过对原始数据的相关矩阵或协方差矩阵进行分析，提取出有代表性的综合指标。

- 区别:

  （1）主成分分析一般不用数学模型来描述,它只是通常的变量变换；因子分析需要构造因子模型(正交或斜交);

  （2）主成分分析中主成分的个数和变量个数p相同,它是将一组具有相关性的变量变换为一组独立的综合变量(注意应用主成分分析解决实际问题时,一般只选取m(m小于p)个主成分)。因子分析的目的是要用尽可能少的公因子,以便构造一个结构简单的因子模型。

  （3）主成分分析是将主成分表示为原变量的线性组合。因子分析是将原始变量表示为公因子和特殊因子的线性组合。

  （4）主成分分析的数学模型本质上是一种线性变换，是将原始坐标变换到变异程度大的方向上去，相当于从空间上转换观看数据的角度，突出数据变异的方向，归纳重要信息。

总的来说，因子分析从本质上看是从显在变量去“提练”潜在因子的过程。正因为因子分析是一个提练潜在因子的过程，因子的个数m取多大是要通过一定规则确定的，并且因子的形式也不是唯一确定的。一般说来，作为“自变量”的因子F1，F2，…，Fm是不可直接观测的。而主成分分析主要通过线性变换将现有变量变成少数几个新的互

不相关的综合变量（主成分），着重解释各变量的总方差，新变量几乎带有原来所有变量的信息；而因子分析则是用少数公共因子与特殊因子来描述原有的变量，更侧重于寻找潜在的因子，并对这些因子进行解释。此外，因子分析中可以对因子进行旋转，使得因子的解释和可读性更强，在解释原始变量的相关性和潜在结构方面更具优势。如果任务是寻找潜在因子并解释数据结构，因子分析更合适；若只是想减少变量数量并保留大部分信息，主成分分析是更好的选择。

5.4.2 因子分析算法评价

（一）优点

- 降低数据维数：因子分析能够将大量的变量归纳为较少数量的因子，有效降低数据的维数，简化数据的解释和分析过程。在市场调研数据分析中，可将众多消费者行为变量归结为几个主要因子，便于理解消费者行为模式。
- 减少噪声和冗余信息：通过识别数据中的共性和模式，因子分析可以减少数据中的噪声和冗余信息，挖掘出数据背后的潜在结构和规律。在基因数据分析中，有助于发现基因之间的潜在关系，去除无关或冗余的基因信息。

（二）缺点

因子分析依赖于对数据的特定假设，如变量之间的线性相关性等。如果这些假设不符合实际情况，可能会导致因子分析结果不准确或失效。在实际应用中，若数据不满足正态分布或变量间相关性并非线性，因子分析的结果可能无法真实反映数据的内在结构。

5.4.3 实践案例：基于因子分析的人格特征潜在因子挖掘

（一）数据准备

本案例选取从国际人格测验项目库中提取的 2800 个受试者的人格测试报告数据。该数据集共有 28 个特征，包含三个人口统计变量（性别、教育和年龄）以及由五类特征组成的 25 个指标（亲和力、尽责性、外向性、神经质和开放性）。

（二）适用性检验

进行因子分析前需要检验特征之间是否具有相关性，如果相关性太低则不适合使用因子分析。常用的适用性检验方法有 KMO 检验和 Bartlett 球性检验。KMO 检验主要测量各变量间的相关性，KMO 取值在 0 到 1 之间，如果 KMO 值大于 0.6，则适合做因子分析；如果该值接近于 1，说明各变量间相关性越强，对其进行因子分析的效果越好。Bartlett 球性检验是关于变量相关系数矩阵的假设检验，检验各个变量是否各自独立。如果 p 值小于显著性水平（一般取 0.05）时，则认为变量之间存在一定的相关性，适合做因子分析。本案例中，求得 p 值小于 0.05，KMO 值为 0.849（大于 0.6），通过适用性检验，可以进一步做因子分析。

（三）模型构建与实现步骤

- 确定公共因子个数：根据原有变量求解协方差矩阵并求解特征值及特征向量，可根据因子特征值大于 1 或累计特征值所占百分比大于 80%，确定公共因子个数。通过碎石图可以看出，有 6 个因子的特征值大于 1，因此选取因子个数为 6。
- 求解初始因子载荷矩阵。
- 进行因子旋转：采用方差最大法将因子载荷矩阵正交旋转，使得因子变量更具有可解释性。
- 计算因子得分：采用最小二乘法计算因子得分。

在 Python 中，FA 算法可以通过调用 factor_analyzer 库的 FactorAnalyzer 类实现，包含计算方差、求解因子载荷矩阵、提取主成分、方差最大旋转、计算因子得分等基本功能。具体代码如下：

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from factor_analyzer import FactorAnalyzer
from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity,calculate_kmo
df = pd.read_csv("D:/3.研究生课程/大数据分析/降维/data/character.csv", index_col=0).reset_index
(drop=True)
#1.数据处理，去掉无效字段与空值
df.drop(["gender","education","age"],axis=1,inplace=True)
df.dropna(inplace=True)
#2.进行适用性检测
chi_square_value, p_value = calculate_bartlett_sphericity(df)
print(f'p值：{p_value}')
kmo_all,kmo_model=calculate_kmo(df)
print(f'KMO：{kmo_model}')
#3.调用因子分析算法拟合数据
fa = FactorAnalyzer(25,rotation='varimax')
fa.fit(df)
#4.求解特征值ev、特征向量v
ev,v=fa.get_eigenvalues()
#5.通过绘制碎石图确定因子个数
plt.rcParams['font.family'] = ['sans-serif']
plt.rcParams['font.sans-serif'] = ['STSong']
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams.update({'font.size': 20})
plt.figure(figsize=(6, 5))
```

```
plt.plot(range(1, df.shape[1] + 1), ev, marker='o', color='#8389E0')
plt.xlabel("因子个数")
plt.ylabel("特征值")
plt.grid()
plt.tight_layout()
plt.show()
#6.选择6个因子构建模型，指定矩阵旋转方式为varimax，实现方差最大化
fa_six = fa = FactorAnalyzer(6,rotation='varimax')
fa_six.fit(df)
#7.求解因子载荷矩阵
fa_six.loadings_
#8.将原始数据用6个因子进行描述
pd.DataFrame(fa_six.loadings_,index=df.columns)
df1=pd.DataFrame(fa_six.transform(df))
df2=pd.DataFrame(fa.get_factor_variance(),index=['variance','proportional_variance','cumulative
_variances'], columns=[f"factor{x}" for x in range(1,7)])
print(round(df1,3))
print(round(df2,3))
```

（四）结果分析

通过因子分析，得到了 6 个新的特征变量来描述原始数据集，将原始数据转成 6 个新的特征。根据因子载荷矩阵可以看出每个因子与原始变量之间的关系，从而对因子进行命名和解释。例如，某个因子在亲和力相关变量上载荷较高，可将其命名为“亲和力因子”。通过这种方式，在保留了原始数据主要信息的情况下，实现了数据的降维，并且能够更好地理解数据的结构和关系。

5.5 多维尺度变换

5.5.1 多维尺度变换基本原理

多维尺度变换（Multidimensional Scaling，MDS）是一种经典的降维方法，其核心目标是在低维空间中重构数据点的相对位置，使得重构后的点之间的距离尽可能接近原始高维空间中数据点之间的距离。简单来说，MDS 试图在保持数据点之间相似性（通常用距离度量）的前提下，将高维数据映射到低维空间，以便于数据的可视化和分析。例如，在社会学研究中，我们可能有关于不同城市之间的经济、文化、人口等多方面的高维数据，通过 MDS 可以将这些城市在二维或三维空间中表示出来，使得城市之间的相对距离反映它们在高维空间中的相似程度。

MDS算法的核心思想是使用距离矩阵来表示数据点之间的相似性或关联性。在实际问题中，我们可能需要分析不同类型的数据，通过将这些数据转换为距离矩阵，我们可以利用MDS算法来挖掘数据中的结构信息和潜在关系。

MDS算法通过将高维数据降维到二维或三维空间，使得数据的可视化分析变得更加直观。此外，在降维过程中，MDS尽量保持数据点之间的距离关系，从而有助于挖掘数据中的真实结构。
在MDS算法中，降维映射的原理是通过保持原始空间中数据点之间的相对距离来将数据从高维空间映射到低维空间。为了便于理解，我们将降维映射过程分为以下几个步骤：

- 构建距离矩阵：首先，我们需要计算原始空间中数据点之间的距离。常用的距离度量方法包括欧几里得距离、Minkowski距离等。通过计算每对数据点之间的距离，我们可以构建一个距离矩阵。
- 中心化距离矩阵：为了进一步处理距离矩阵，我们需要对其进行中心化处理，使得数据点相对于原点对称。
- 计算内积矩阵：通过中心化距离矩阵，我们可以计算内积矩阵B。内积矩阵表示数据点之间的内积关系，可以用于进一步分析数据的结构。
- 计算特征值和特征向量：在得到内积矩阵B后，我们需要计算其特征值和特征向量。特征值表示数据的主要变化方向，特征向量表示对应方向上的大小。我们将选取最大的k个特征值及其对应的特征向量，作为降维后的k维空间的基。
- 计算降维后的坐标：将原始数据投影到选定的k维基上，我们可以得到降维后的坐标。具体计算方法为：新坐标 = 特征向量矩阵 * 特征值矩阵的平方根。这样，我们就得到了降维后的数据表示。

在MDS（多维尺度分析）中，存在两种主要的度量方式：度量 MDS 和非度量 MDS。

1. 度量 MDS（Metric MDS / Classical MDS）：这种方法旨在保持原始数据中的点之间的欧几里德距离或其他度量距离在低维嵌入空间中的相对关系。它试图在嵌入空间中尽可能准确地保持原始数据点之间的距离。度量 MDS 在优化过程中尝试最小化原始距离与嵌入距离之间的误差。
2. 非度量 MDS（Non-Metric MDS）：在这种情况下，我们只知道原始数据点之间的相对排序关系，而不知道确切的距离或度量。非度量 MDS 通过保持点的排序关系来在低维空间中嵌入数据。它使用的度量是一种序关系度量，例如排名。

因此，在MDS中，“metric” 参数用于指定使用哪种类型的度量方式。如果设置为True，表示使用度量 MDS，而如果设置为False，表示使用非度量 MDS。度量方式的选择会影响嵌入结果和优化过程。
MDS的python实现：
在sklearn机器学习包中，练习MDS的相关代码。

```
import numpy as np
import pandas as pd
from sklearn.manifold import MDS
import matplotlib.pyplot as plt


data = np.array([(5,0,0,0,0,1,0,2,1,0),
                 (0,0,3,0,3,0,1,0,0,1),
```

```
                    (2,0,0,0,0,0,1,0,0,0),
                    (1,0,1,0,2,0,0,0,0,1),
                    (5,0,2,0,0,4,2,2,3,7),
                    (0,3,0,1,0,0,0,0,0,0),
                    (0,0,0,6,0,0,0,0,0,1),
                    (0,5,0,0,0,0,0,0,0,0),
                    (0,1,0,0,0,0,0,0,0,0),
                    (0,2,0,0,0,0,0,0,0,0)
                   ]
                  )
index = ['auto1','auto2','auto3','auto4','auto5','moto1','moto2','moto3','moto4','moto5']
columns = ['car','bike','cars','his','tires','she','ive','her','#k','are']
Word = pd.DataFrame(data,index,columns)

mds = MDS()
mds.fit(data)
a = mds.embedding_
plt.scatter(a[0:5,0],a[0:5,1],color='turquoise')
plt.scatter(a[5:10,0],a[5:10,1],color='red')
```

我们接下来再用一个简单的例子，通过将3维空间的正方体的6个顶点的数据应用MDS进行二维和一维空间映射，巩固MDS的使用。

1. 导入相关的包：

```
from sklearn.manifold import MDS
from matplotlib import pyplot as plt
import numpy as np
```

2. 下面代码创建MDS对象进行分别进行2维和1维训练fit，返回在2维和1维空间中的嵌入点。

```
X = np.array([[0, 0, 0], [0, 0, 1], [1, 1, 1], [0, 1, 0], [0, 1, 1],[1, 1, 0]])
mds = MDS(random_state=0)
X_transform = mds.fit_transform(X)
print(X_transform)

mds_1 = MDS(n_components=1,random_state=0)
```

```
X_transform_1 = mds_1.fit_transform(X)
print(X_transform_1)
```

3. MDS默认嵌入是基于应力最小化算法(SMACOF)创建的，因此输出stress(拟合优度统计量)变量查看拟合程度：

```
print('2D stress:',mds.stress_)
print('1D stress:',mds_1.stress_)
```

4. 可视化对比 数据点在 3D、2D、1D空间的位置：上述过程并没有帮助我们很好地理解发生了什么，人类在处理数字方面并不是很擅长；为了更好地理解整个过程，绘制原始点及其通过MDS(欧几里德距离)训练返回的嵌入点。原始点和其对应的嵌入点使用相同的颜色进行显示：

```
colors =  ['r', 'g', 'b', 'c', 'm', 'y']
size = [64, 64, 64, 64, 64,64]
fig = plt.figure(figsize=(14,6.5))
ax = fig.add_subplot(1,2,1, projection='3d')
#ax.set_position([0.2,0.2,2, 2])

plt.scatter(X[:,0], X[:,1], zs=X[:,2], s=size, c=colors)
plt.title('Original Points(3D)')

ax = fig.add_subplot(2,2,2)
plt.scatter(X_transform[:,0], X_transform[:,1], s=size, c=colors)
plt.title('Embedding in 2D')

ax = fig.add_subplot(2,2,4)
plt.scatter(X_transform_1[:,0],[0,0,0,0,0,0],s=size, c=colors)
plt.title('Embedding in 1D')

fig.subplots_adjust(wspace=.4, hspace=0.5)
plt.show()
```

使用MDS 的另一种方法是构建距离矩阵并将 MDS 直接应用于该矩阵，如下面的代码所示。当需要欧氏距离以外的距离测量时，此方法非常有用。下面的代码计算成对的曼哈顿距离（也称为城市街区距离或 L1 距离）并通过 MDS 转换数据。

dissimilarity参数已设置为precomputed：

```
from sklearn.metrics.pairwise import manhattan_distances, euclidean_distances


print('原始3维数据:\n',X)
dist_manhattan = manhattan_distances(X)
print('曼哈顿距离矩阵:\n',dist_manhattan)
mds = MDS(dissimilarity='precomputed', random_state=0)
# Get the embeddings
X_transform_L1 = mds.fit_transform(dist_manhattan)
print('MDS 降维后的嵌入点:\n',X_transform_L1)
print('stress:',mds.stress_)
```

5.5.2 多维尺度变换算法评价

多维尺度分析（MDS）是一种用于降维的非线性技术，通过将高维数据映射到低维空间来保留数据模式。在机器学习和数据分析中，MDS广泛应用于简化数据并保持数据点间的距离关系。度量MDS保持欧几里得距离，非度量MDS维护排名距离。Scikit-Learn的sklearn.manifold模块支持MDS，通过调整参数如维度数、度量类型，可以灵活地实现数据降维。使用MDS对Olivetti Faces数据进行降维展示了不同维度的效果，寻找最佳维度可平衡信息保留与降维。

（一）优点

- 保留全局结构：MDS 能够很好地保留数据点之间的全局距离信息，使得在低维空间中数据点的相对位置关系与高维空间中尽量一致。这有助于我们从整体上把握数据的分布和结构，发现数据中的潜在模式和规律。
- 简单直观：MDS 的原理和实现过程相对简单，易于理解和解释。它不依赖于复杂的模型假设，只需要输入数据点之间的距离矩阵，就可以进行降维和可视化。
  无需参数调整：与一些复杂的降维算法相比，MDS 不需要调整大量的参数，使用起来比较方便。只需要确定要降维到的目标维度即可。

（二）缺点

- 线性假设限制：MDS 基于线性变换的思想，假设数据在高维空间中的结构可以通过线性映射在低维空间中近似表示。然而，实际数据往往具有复杂的非线性结构，当数据的非线性特征较强时，MDS 可能无法准确地还原数据的真实结构，导致降维效果不佳。
- 计算复杂度高：对于大规模数据集，计算距离矩阵和进行特征值分解的计算量较大，时间和空间复杂度都较高。特别是当样本数量 很大时，距离矩阵的存储和计算会成为瓶颈。
- 对异常值敏感：MDS 依赖于数据点之间的距离信息，异常值的存在会显著影响距离矩阵的计算，进而影响降维结果。一个异常值可能会导致整个数据的分布在低维空间中发生扭曲。

（三）多维尺度变换的实际应用案例

由于对多维尺度变换比较陌生，我去了解了一下多维尺度变换在实际生活中的应用例子。

1. 市场研究领域

   在市场研究中，企业常常需要了解消费者对不同品牌产品的认知和偏好。通过设计调查问卷，收集消费者对多个品牌在多个属性（如价格、质量、外观、功能等）上的评价数据。将每个品牌看作一个数据点，根据消费者的评价计算品牌之间的相似度（距离）矩阵，然后使用 MDS 进行降维。

   例如，一家化妆品公司想要了解旗下不同系列产品在消费者心中的位置。通过调查收集了消费者对 10 个不同系列化妆品在保湿效果、美白效果、控油效果、包装设计、价格等 8 个属性上的评分。计算这 10 个系列产品之间的欧氏距离矩阵，再利用 MDS 将其降维到二维空间。在二维可视化图中，位置相近的系列产品表示在消费者认知中具有较高的相似度，企业可以根据这个结果调整产品定位和营销策略。如果发现某些系列产品过于集中，说明市场定位存在重叠，可考虑对产品进行差异化改进；如果某些系列产品与其他产品距离较远，可分析其独特卖点并加大宣传力度。

2. 生物学领域

   在生物学中，MDS 可用于分析物种之间的进化关系。生物学家通过测量多个物种在基因序列、形态特征等多个方面的数据，计算物种之间的遗传距离或形态距离。以基因序列分析为例，对不同物种的特定基因片段进行测序，根据碱基对的差异计算物种之间的遗传距离，构建距离矩阵。

   比如，研究人员对 20 种鸟类的线粒体 DNA 序列进行分析，计算它们之间的遗传距离。利用 MDS 将这些鸟类的高维基因数据降维到二维或三维空间，绘制进化关系图。在图中，距离较近的鸟类表示它们在进化上的亲缘关系较近，可能具有共同的祖先；距离较远的鸟类则亲缘关系较远。这有助于生物学家理解物种的进化历程、分类关系以及生物多样性的形成机制。

3. 心理学领域

   在心理学研究中，MDS 可用于分析人类对不同刺激的感知和认知结构。心理学家通过让受试者对一系列刺激（如颜色、声音、图形等）进行相似度判断，收集主观相似度数据，构建距离矩阵。

   例如，在颜色感知研究中，让受试者比较不同颜色之间的相似程度，根据受试者的判断构建颜色之间的距离矩阵。使用 MDS 对这些颜色进行降维并可视化。结果可能会呈现出与人类颜色视觉理论相符的结构，如在二维图中形成颜色环，反映出颜色之间的色相、饱和度和明度等关系。这有助于心理学家深入了解人类的感知机制和认知过程。

4. 信息检索领域

   在信息检索中，MDS 可以用于文档聚类和可视化。将文档集合中的每个文档看作一个数据点，通过计算文档之间的相似度（如基于词频 - 逆文档频率，TF - IDF）构建距离矩阵。

   比如，在一个新闻文档集合中，有 500 篇关于不同主题的新闻报道。计算这些文档之间的 TF - IDF 相似度矩阵，使用 MDS 将文档降维到二维空间。在可视化图中，相似主题的文档会聚集在一起，形成不同的簇。用户可以直观地看到文档集合的整体结构，快速找到感兴趣的文档簇，提高信息检索的效率。同时，文档聚类结果也可以为搜索引擎的分类和推荐提供依据。

5.5.3 实践案例：基于多维尺度变换的人脸数据集降维

Scikit-Learn 库的sklearn.manifold模块实现了流形学习和数据嵌入技术,包括MDS类(class)。嵌入(embeddings)是通过使用主化应力最小化 (SMACOF)算法来确定的。MDS对象的常用重要参数包括：

- n_components：将点映射到的维度数。默认值为 2。
  MDS 涉及的重要超参数之一是嵌入点的低维空间的维度数。当 MDS 用作降维的预处理步骤时，这将非常相关。问题是：选择多少个维度？在不丢失重要信息的情况下最大限度地降低维度？简单方法是对不同的n_components值运行 MDS并计算出拟合优度统计量stress_。stress_值随着维度的增加而减小:选择一个在 n_components和stress_之间进行公平权衡的点。
- metric:布尔变量,默认值：True，“metric” 指的是度量方式或者度量类型。
- dissimilarity：默认值为euclidean，指定欧氏成对距离。另一个可能的值是precomputed. 使用precomputed需要计算成对距离矩阵并使用该矩阵作为fit() or fit_transform()函数的输入。

与MDS对象相关联的四个属性（训练完成的主要返回对象/属性）为：

- embedding_：新空间中点的位置数据。
- stress_：MDS 中使用的拟合优度统计量。
- dissimilarity_matrix_：成对距离/相异度矩阵。
- n_iter_：与最佳拟合优度测量相关的迭代次数。

与用于机器学习的scikit-learn其他类一样，MDS也实现fit()和fit_transform()方法。
Olivetti Faces(人脸数据库)是一个常用的计算机视觉和人脸识别领域的数据集，由Olivetti研究公司创建。该数据库包含了一组40个志愿者的人脸图像，每个志愿者提供了10张不同表情和光照条件下的人脸照片。因此，总共有400张 64x64 位图人脸图像。多维尺度分析（MDS）对人脸数据进行降维度，同时保持数据中的模式，使得同一人的不同人脸图像在二维空间中彼此靠近，而与另一个人映射的人脸图像保持一定的距离。接下来用如下代码对人脸数据集进行降维：

- 引入数据
- 编写输出函数mapData：该函数将 MDS 应用于距离矩阵，并在 2D 空间中显示变换后的点，相同颜色的点表示同一个人的映射图像。在第二张图中，它还显示了图表上每个人脸的图像及不同人显示不同的边框颜色，并将其映射到低维空间中。

```
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import patches
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
from sklearn.manifold import MDS
```

```
from sklearn.metrics import euclidean_distances
from sklearn.datasets import fetch_olivetti_faces as fof

# 1.读取数据
faces = fof()
X_faces = faces.data
y_faces = faces.target
ind = y_faces < 5
X_faces = X_faces[ind, :]
y_faces = y_faces[ind]

plt.figure(figsize=(20, 10))
for i in range(50):  # 显示前50张面孔
    plt.subplot(5, 10, i + 1)
    plt.imshow(X_faces[i].reshape(64, 64), cmap='gray')
    plt.axis('off')
plt.subplots_adjust(wspace=0.2, hspace=0.2)
plt.show()

# 2.调用MDS算法对距离矩阵降维
def mapData(dist_matrix, X, y, metric, title):
    mds = MDS(metric=metric, dissimilarity='precomputed', random_state=0)
    pts = mds.fit_transform(dist_matrix)

    fig, ax = plt.subplots(figsize=(6, 5))
    color = ['r', 'g', 'b', 'c', 'm']
    sns.scatterplot(x=pts[:, 0], y=pts[:, 1], hue=y, palette=color, ax=ax)

    for x, ind in zip(X[1:], range(1, pts.shape[0])):
        im = x.reshape(64, 64)
        imagebox = OffsetImage(im, zoom=0.3, cmap=plt.cm.gray)
        i = pts[ind, 0]
        j = pts[ind, 1]
        ab = AnnotationBbox(imagebox, (i, j), frameon=False)
        ax.add_artist(ab)
        renderer = fig.canvas.get_renderer()
        bbox = ab.get_window_extent(renderer=renderer)
```

```
        w = (max(pts[:, 0]) - min(pts[:, 0])) / bbox.width
        h = (max(pts[:, 1]) - min(pts[:, 1])) / bbox.height
        rect = patches.Rectangle((i - w, j - h), w * 2, h * 2,
                                 edgecolor=color[y[ind]], linewidth=1, fill=False)
        ax.add_patch(rect)

    ax.legend(fontsize='large', title_fontsize='large')
    plt.rcParams['font.family'] = ['sans-serif']
    plt.rcParams['font.sans-serif'] = ['Times New Roman']
    plt.rcParams.update({'font.size': 20})
    plt.tick_params(axis='both', which='major', labelsize=20)
    plt.savefig('mds1.png', dpi=600)
    plt.show()

# 3.计算欧氏距离矩阵
dist_euclid = euclidean_distances(X_faces)
mapData(dist_euclid, X_faces, y_faces, True, 'MDS')
```

结果分析：

通过上述代码，我们将城市的高维数据降维到二维空间，并进行了可视化。从可视化结果中，我们可以直观地看到不同城市之间的相对位置关系。如果两个城市在二维空间中的距离较近，说明它们在高维空间中的相似度较高；反之，如果距离较远，则相似度较低。这样，我们可以根据降维结果对城市进行聚类分析或其他进一步的研究。

同时，我们也可以根据实际情况调整降维的目标维度，以更好地展示数据的结构。例如，如果二维空间无法很好地展示数据的特征，我们可以尝试将数据降维到三维空间进行可视化。

5.6 等距特征映射

5.6.1 等距特征映射基本原理

（一）基本思想及涉及概念

等距特征映射（Isometric Mapping，IsoMap）是一种基于流形学习的非线性降维算法，旨在解决高维数据在降维过程中保持数据点之间真实几何关系的问题。其核心思想基于对传统线性降维方法局限性的改进。在处理非线性结构的数据时，传统方法如欧式距离在高维空间中无法准确反映数据点之间的真实距离关系，而 IsoMap 算法通过引入流形中的测地距离来表示高维空间的距离，然后借助多维尺度变换（MDS）算法进行降维。Isomap 是对经典多维缩放（MDS）的扩展，通过保留点对点之间的流形距离来实现降维。

高维空间中的数据通常位于低维流形上（如曲面或曲线）。在高维空间中，欧几里得距离可能无法准确反映数据

的真实结构，而沿流形的距离（即流形距离）能够更好地描述数据点之间的关系。

Isomap 的目标是：

（1）计算高维数据点之间的流形距离；

（2）使用这些流形距离在低维空间中重构数据点的位置。

接着，我们学习一下关于流形学习的概念，我们先来理解流形和流形距离： 流形(manifold)是一般几何对象的总称，包括各种维度的曲线与曲面等。流形距离（Manifold Distance）则是数据点沿其所在流形表面之间的最短路径距离。它区别于欧几里得距离，后者是在原始高维空间中测量的直线距离，而流形距离反映了数据在低维流形结构上的真实几何关系。例如二维曲面上，欧几里得距离只能表示直线距离，而不能表示真实几何关系（曲面）。和一般的降维分析一样，流形学习是把一组在高维空间中的数据在低维空间中重新表示。不同之处是，在流形学习中假设：所处理的数据采样与一个潜在的流形上，或者说对于这组数据存在一个潜在的流形。而流形学习（manifold learning）是一类借鉴了拓扑流行概念的降维方法，在降维时，若低维流行嵌入到高维空间中，则数据样本在高维空间的分布虽然看上去十分复杂，但在局部上仍具有欧式空间（对现实空间的规则抽象和推广）的性质。

- 为什么要用流行学习降维：我所能观察到的数据其实是由一个低维流形映射到高维空间上的（一个立方体可以展开为平面）。由于数据内部特征的限制，一些高维中的数据会产生维度上的冗余，实际上只需要比较低的维度就能唯一的表示。

通俗的说，流行学习可以概括为：在保持流形上点的某些几何性质特征（圆形、瑞士卷形）的情况下，找出一组对应的内蕴坐标，将流形尽量好的展开在低维平面上。举个例子帮助理解：当我们拿到一个地球仪，然后想要知道重庆到伦敦的距离，我们不能直接在地球仪内部用直尺来测量空间上的距离，而要想办法将地球仪展为平面（降维），再进行测量距离（测地线=>本真距离）。

说到实际应用，以三维空间中的瑞士卷（Swiss roll）数据集为例，在低维嵌入流形上，数据点之间的真实距离是测地线距离（如曲面上两点间的最短路径），而非欧式距离（直线距离）。IsoMap 算法通过计算样本之间的距离矩阵，构建近邻图，利用最短路径算法计算邻接图中数据点之间的最短路径，将计算两点之间测地线距离的问题转变为寻找近邻连接图上两点之间的最短路径问题。在得到测地距离矩阵后，再通过 MDS 算法对数据进行降维，从而在低维空间中还原出数据集的真实低维流形结构，使得相似的近邻点在低维空间中靠近，更好地保留数据的全局结构和流形特性。

对于我们的示例，让我们创建一个称为瑞士卷的 3D 对象。该对象由 2,000 个单独的数据点组成。接下来，我们要使用 Isomap 将这个 3 维瑞士卷映射到 2 维。 要跟踪此转换过程中发生的情况，让我们选择两个点：A 和 B。我们可以看到这两个点在 3D 空间内彼此相对靠近。 如果我们使用诸如 PCA 之类的线性降维方法，那么这两个点之间的欧几里得距离在较低维度上会保持一些相似。 请注意，PCA 中 2D 对象的形状看起来像是从特定角度拍摄的同一 3D 对象的图片。这是线性变换的一个特点。同时，诸如 Isomap 之类的非线性方法给了我们非常不同的结果。 我们可以将这种转换描述为展开瑞士卷并将其平放在 2D 表面上。我们可以看到，二维空间中点 A 和 B 之间的距离基于通过邻域连接计算的测地线距离。这就是 Isomap 能够执行非线性降维的秘诀，它专注于保留局部结构而较少关注全局结构。

接下来利用scikit-learn提供现成的模块对三维空间中的瑞士卷进行代码实现：

```
from sklearn.manifold import Isomap
from sklearn.datasets import make_swiss_roll
import matplotlib.pyplot as plt

# 生成三维的瑞士卷数据集
X, color = make_swiss_roll(n_samples=1000, noise=0.05)

# 可视化原始高维数据
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=color, cmap=plt.cm.Spectral)
plt.title("Original 3D Swiss Roll")
plt.show()



# 设置 参数
n_neighbors = 10  # 每个点的邻居数量
n_components = 2  # 降维后的目标维度

# 创建 Isomap 模型
isomap = Isomap(n_neighbors=n_neighbors, n_components=n_components)
# 降维
X_isomap = isomap.fit_transform(X)

# 可视化降维后的结果
plt.figure(figsize=(8, 6))
plt.scatter(X_isomap[:, 0], X_isomap[:, 1], c=color, cmap=plt.cm.Spectral)
plt.title("2D Embedding using Isomap")
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.show()
```

（二）等距映射（Isomap）的步骤

- 建图

  流形在局部上与欧几里得空间同胚（定义，局部欧几里得性），所以我们可以对每个点基于欧氏距

离找出近邻点，然后建立邻接图，边权为两点间的欧几里得距离。

其中近邻图可以通过两种方法确定，包括指定近邻点的个数，比如欧氏距离最近的k个点构成的k近邻图；还有指定距离阈值，距离小于阈值的点构成近邻图。

- 寻找最短路和最短流形距离

  在图上跑最短路算法（Floyd，Dijkstra，Bellman-Ford，SPFA），即可找到两个点之间的最短路程，这个路程被称为最短流形距离。

- 多维缩放（MDS）

  在最短流形距离的基础上进行经典MDS。

  多维缩放是一种经典的降维方法，旨在从高维数据中提取低维嵌入，同时尽可能保留数据点之间的距离关系。MDS 的核心目标是将高维点的距离关系映射到低维空间中，使得低维空间中的点对之间的距离与原始距离尽可能相似。它有两种求解过程：梯度下降和特征值分解。

通过这些步骤，Isomap可以有效地将数据降维，同时保持数据的全局几何结构。
总的来说，高维空间的数据大多数都具有相同的特点，因此会嵌入到一个流行体中，而不是随机分散在高维空间中，高维数据相同特征的数目也就是流行体的维度，也就是我们降维的目标空间维数，这个维度需要一定的人为主观判断。同时，如果数据在高维空间中没有嵌入到一个流行体中。例如高维数据点分为10类，每一类都和其它类分散开，自己成为一簇，此时isomap算法就不太适合。如果数据是分类的，数据基本不会嵌入在一个流行体，isomap降维算法效果就比较差，但是数据是连续的，数据就很有可能嵌入在一个流行空间内，此时isomap算法的效果就会比较好。
为了更好运用Isomap算法，现在让我们使用 Isomap 来降低 MNIST 数据集（手写数字集合）中图片的高维数。这将使我们能够看到不同的数字如何在 3D 空间中聚集在一起。

- 设置

  我们将使用以下数据和库：

  1. Scikit-learn
  2. Plotly 和 Matplotlib
  3. Pandas

- 让我们导入库。

```
import pandas as pd # for data manipulation
 # Visualization
 import plotly.express as px # for data visualization
 import matplotlib.pyplot as plt # for showing handwritten digits


# Skleran
from sklearn.datasets import load_digits # for MNIST data
from sklearn.manifold import Isomap # for Isomap dimensionality reduction
```

- 接下来，我们加载 MNIST 数据。

```
# Load digits data
digits = load_digits()

# Load arrays containing digit data (64 pixels per image) and their true labels
X, y = load_digits(return_X_y=True)

# Some stats
print('Shape of digit images: ', digits.images.shape)
print('Shape of X (training data): ', X.shape)
print('Shape of y (true labels): ', y.shape)
```

- 让我们显示前 10 个手写数字，以便更好地了解我们正在处理的内容。

```
# Display images of the first 10 digits
fig, axs = plt.subplots(2, 5, sharey=False, tight_layout=True, figsize=(12,6), facecolor='whit
e')
n=0
plt.gray()
for i in range(0,2):
    for j in range(0,5):
        axs[i,j].matshow(digits.images[n])
        axs[i,j].set(title=y[n])
        n=n+1
plt.show()
```

- 我们现在将应用 Isomap 将 X 数组中每条记录的维数从 64 减少到 3。

```
### Step 1 - Configure the Isomap function, note we use default hyperparameter values in this e
xample
embed3 = Isomap(
    n_neighbors=5, # default=5, algorithm finds local structures based on the nearest neighbors
    n_components=3, # number of dimensions
    eigen_solver='auto', # {‘auto’, ‘arpack’, ‘dense’}, default=’auto’
    tol=0, # default=0, Convergence tolerance passed to arpack or lobpcg. not used if eigen_sol
ver == ‘dense’.
```

```
    max_iter=None, # default=None, Maximum number of iterations for the arpack solver. not used if eigen_solver == 'dense'.
    path_method='auto', # {'auto', 'FW', 'D'}, default='auto', Method to use in finding shortest path.
    neighbors_algorithm='auto', # neighbors_algorithm{'auto', 'brute', 'kd_tree', 'ball_tree'}, default='auto'
    n_jobs=-1, # n_jobsint or None, default=None, The number of parallel jobs to run. -1 means using all processors
    metric='minkowski', # string, or callable, default="minkowski"
    p=2, # default=2, Parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2
    metric_params=None # default=None, Additional keyword arguments for the metric function.
)

### Step 2 - Fit the data and transform it, so we have 3 dimensions instead of 64
X_trans3 = embed3.fit_transform(X)

### Step 3 - Print shape to test
print('The new shape of X: ',X_trans3.shape)
```

- 最后，让我们绘制一个 3D 散点图，看看将维度降低到 3 后数据的样子。

```
# Create a 3D scatter plot
fig = px.scatter_3d(None,
                    x=X_trans3[:,0], y=X_trans3[:,1], z=X_trans3[:,2],
                    color=y.astype(str),
                    height=900, width=900
                   )

# Update chart looks
fig.update_layout(#title_text="Scatter 3D Plot",
                  showlegend=True,
                  legend=dict(orientation="h", yanchor="top", y=0, xanchor="center", x=0.5),
                  scene_camera=dict(up=dict(x=0, y=0, z=1),
                                    center=dict(x=0, y=0, z=-0.2),
                                    eye=dict(x=-1.5, y=1.5, z=0.5)),
                                    margin=dict(l=0, r=0, b=0, t=0),
```

```
                  scene = dict(xaxis=dict(backgroundcolor='white',
                                          color='black',
                                          gridcolor='#f0f0f0',
                                          title_font=dict(size=10),
                                          tickfont=dict(size=10),
                                         ),
                               yaxis=dict(backgroundcolor='white',
                                          color='black',
                                          gridcolor='#f0f0f0',
                                          title_font=dict(size=10),
                                          tickfont=dict(size=10),
                                          ),
                               zaxis=dict(backgroundcolor='lightgrey',
                                          color='black',
                                          gridcolor='#f0f0f0',
                                          title_font=dict(size=10),
                                          tickfont=dict(size=10),
                                         )))

# Update marker size
fig.update_traces(marker=dict(size=2))

fig.show()
```

Isomap 在将维度从 64 减少到 3 方面做得非常出色，同时保留了非线性关系。 这使我们能够在 3 维空间中可视化手写数字的簇。

Isomap 是降维的最佳工具之一，使我们能够保留数据点之间的非线性关系。我们已经看到了 Isomap 算法如何在实践中用于手写数字识别。 同样，我们可以使用 Isomap 作为 NLP（自然语言处理）分析的一部分，以在训练分类模型之前减少文本数据的高维。

5.6.2 等距特征映射算法评价

（一）优点

- 有效处理非线性数据：IsoMap 算法能够较好地处理具有非线性结构的数据，通过测地距离和流形学习的方法，保留数据的全局结构和流形特性，在降维的同时尽量保持原始数据之间的相对位置关系，这是其相较于传统线性降维方法的显著优势。

- 还原真实低维结构：该算法可以在较低维数下对数据进行可视化展示，帮助用户直观地理解数据的内在结构和分布规律。在处理地理空间数据、生物数据等具有复杂非线性结构的数据时，IsoMap 算法能够还原出数据的真实低维流形结构，为数据分析和挖掘提供有力支持。

（二）缺点

- 计算复杂度高：IsoMap 算法需要计算所有点之间的距离，构建近邻图并计算最短路径，计算复杂度较高。尤其是在处理大规模数据集时，计算量会显著增加，导致算法运行时间较长，对计算资源的要求也较高。
- 空间复杂度高：在计算和存储距离矩阵、近邻图以及中间计算结果时，需要占用大量的内存空间，空间复杂度较高。这对于内存资源有限的系统来说，可能会成为限制其应用的因素。
- 对近邻参数敏感：近邻图的构建依赖于近邻点个数或距离阈值的选择，不同的参数设置可能会导致不同的降维结果。如果参数选择不当，可能无法准确反映数据的真实结构，影响降维效果。

5.6.3 实践案例：基于等距特征映射的S状流形模型降维

（一）数据准备与分析

在 Python 中，通过调用 Scikit - learn 库中 datasets 模块的 make_s_curve 函数生成 S 状流形数据集。该数据集是一个经典的二维流形数据集，在三维空间中具有螺旋状的结构，可用于测试和验证 IsoMap 算法的性能。之后使用 Scikit - learn 库中的 manifold 模块的 IsoMap 函数实现 IsoMap 算法。首先创建 IsoMap 对象并设置参数，如近邻点个数、降维后的维数等，然后调用 fit_transform 方法拟合数据并进行降维。

代码：

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_s_curve
from sklearn.manifold import Isomap
from matplotlib.ticker import MaxNLocator
# 1.构建数据集
n_points = 5000
X, color = make_s_curve(n_points, random_state=0)
n_neighbors = 30
n_components = 2
plt.rcParams['font.sans-serif']=['Times New Roman']
plt.rcParams.update({'font.size': 35})
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(121, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=color, cmap=plt.cm.Spectral)
```

```
ax.view_init(4, -72)
ax.tick_params(axis='both', which='major')  # Increase axis label size
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
ax.yaxis.set_major_locator(MaxNLocator(integer=True))
ax.zaxis.set_major_locator(MaxNLocator(integer=True))
# 2.设置近邻点个数为50，将样本点映射至二维空间中
Y = Isomap(n_neighbors=n_neighbors, n_components=n_components).fit_transform(X)
ax2 = fig.add_subplot(122)
scatter = ax2.scatter(Y[:, 0], Y[:, 1], c=color, cmap=plt.cm.Spectral)
ax2.tick_params(axis='both', which='major')
ax2.xaxis.set_major_locator(MaxNLocator(integer=True))
ax2.yaxis.set_major_locator(MaxNLocator(integer=True))
plt.show()
```

（二）结果分析

通过上述代码，对 S 状流形数据集进行降维分析。从降维后的可视化结果可以看出，IsoMap 算法能够有效地将三维的 S 状流形数据降维到二维空间，并且在降维过程中较好地保留了数据的结构特征。相似的近邻点在二维空间中靠近，不同区域的数据点分布与原始三维数据的结构相对应，使得数据的内在结构更加清晰直观。这表明 IsoMap 算法在处理具有复杂非线性结构的数据时具有较好的性能，可以为后续的数据分析和处理提供有效的支持。同时，也可以通过调整 IsoMap 算法的参数，如近邻点个数、降维维数等，观察降维结果的变化，以选择最优的参数设置。

5.7 线性判别分析

5.7.1 线性判别分析基本原理

（一）基本思想

线性判别分析（Linear Discriminant Analysis，LDA）是一种有监督的线性降维算法，其核心目标是在降维过程中最大化类间距离，同时最小化类内距离，从而实现不同类别数据在低维空间中的有效分离。简单来说，LDA 旨在找到原始数据特征变量的一个线性组合，将数据投影到一个新的空间，使得不同类别数据投影点的中心尽可能远，同类别数据的投影点尽可能接近。例如在图像识别领域，对于不同类别的图像（如猫和狗的图像），LDA 可以找到一种投影方式，将高维的图像特征投影到低维空间，使得猫和狗的图像在低维空间中能够清晰地分开，便于后续的分类识别。

线性判别分析 (LDA)是对费舍尔的线性鉴别方法(FLD)的归纳，属于监督学习的方法。LDA使用统计学，模式识别和机器学习方法，试图找到两类物体或事件的特征的一个线性组合，以能够特征化或区分它们。所得的组合可用来作为一个线性分类器，或者，更常见的是，为后续的分类做降维处理。

LDA的核心思想是：类内小，类间大。该如何理解呢？所谓的“类内小，类间大”，就是，投影后类内方差最小，

类间方差最大。将二维数据，在一维进行投影，投影后希望每一种类别数据的投影点尽可能的接近，而不同类别的数据的类别中心之间的距离尽可能的大。这样就可以很好地区分两类，进而分割。实际上，通过上面的过程也可以知道，通过LDA也可以进行数据的降维。

根据以上基本思想，我们需要将样本点投影到最佳鉴别矢量空间，达到以下两个要求：最大化类间距离与最小化类中样本方差。

（二）实现步骤

- 计算类内散度矩阵与类间散度矩阵：根据数据集的类别信息，分别计算每个类别的数据均值和协方差。
- 构造最大化目标函数：构建目标函数，以确定最佳投影方向。
- 求解投影方向：通过求解广义特征值问题，得到特征值和特征向量，选择合适的特征向量作为投影方向。在多类 LDA 中，通常选择前个最大特征值所对应的特征向量构成投影矩阵。
- 计算投影后的数据点：根据得到的投影方向，计算投影后的数据点，将原始数据投影到新的低维空间。

（三）对比PCA

LDA用于降维，和PCA有很多相同，也有很多不同的地方，因此值得好好的比较一下两者的降维异同点。

- 相同点：
    1. 两者均可以对数据进行降维。
    2. 两者在降维时均使用了矩阵特征分解的思想。
    3. 两者都假设数据符合高斯分布。
- 不同点：
    1. LDA是有监督的降维方法，而PCA是无监督的降维方法
    2. LDA降维最多降到类别数k-1的维数，而PCA没有这个限制。
    3. LDA除了可以用于降维，还可以用于分类。
    4. LDA选择分类性能最好的投影方向，而PCA选择样本点投影具有最大方差的方向。在某些数据分布下LDA比PCA降维较优。

接下来，我们使用scikit-learn实现LDA的代码熟悉相关操作，这里直接通过scikit-learn的接口来生成数据：

```
from sklearn.datasets import make_multilabel_classification
import numpy as np

x, y = make_multilabel_classification(n_samples=20, n_features=2,
                                      n_labels=1, n_classes=1,
                                      random_state=2)  # 设置随机数种子，保证每次产生相同的数据。

# 根据类别分个类
```

```
index1 = np.array([index for (index, value) in enumerate(y) if value == 0])  # 获取类别1的index
s
index2 = np.array([index for (index, value) in enumerate(y) if value == 1])  # 获取类别2的index
s

c_1 = x[index1]  # 类别1的所有数据(x1, x2) in X_1
c_2 = x[index2]  # 类别2的所有数据(x1, x2) in X_2
#print(c_1)
#print(c_2)
```

之后，进行LDA算法实现：

```
def cal_cov_and_avg(samples):
    """
    给定一个类别的数据，计算协方差矩阵和平均向量
    :param samples:
    :return:
    """
    u1 = np.mean(samples, axis=0)
    cov_m = np.zeros((samples.shape[1], samples.shape[1]))
    for s in samples:
        t = s - u1
        cov_m += t * t.reshape(2, 1)
    return cov_m, u1


def fisher(c_1, c_2):
    """
    fisher算法实现
    :param c_1:
    :param c_2:
    :return:
    """
    cov_1, u1 = cal_cov_and_avg(c_1)
    cov_2, u2 = cal_cov_and_avg(c_2)
    s_w = cov_1 + cov_2
```

```
    u, s, v = np.linalg.svd(s_w)  # 奇异值分解
    s_w_inv = np.dot(np.dot(v.T, np.linalg.inv(np.diag(s))), u.T)
    return np.dot(s_w_inv, u1 - u2)
```

然后进行判定类别：

```
def judge(sample, w, c_1, c_2):
    """
    true 属于1
    false 属于2
    :param sample:
    :param w:
    :param center_1:
    :param center_2:
    :return:
    """
    u1 = np.mean(c_1, axis=0)
    u2 = np.mean(c_2, axis=0)
    center_1 = np.dot(w.T, u1)
    center_2 = np.dot(w.T, u2)
    pos = np.dot(w.T, sample)
    return abs(pos - center_1) < abs(pos - center_2)



w = fisher(c_1, c_2)  # 调用函数，得到参数w
out = judge(c_1[1], w, c_1, c_2)   # 判断所属的类别
print(out)
```

最后绘图：

```
import matplotlib.pyplot as plt


plt.scatter(c_1[:, 0], c_1[:, 1], c='#99CC99')
plt.scatter(c_2[:, 0], c_2[:, 1], c='#FFCC00')
line_x = np.arange(min(np.min(c_1[:, 0]), np.min(c_2[:, 0])),
                   max(np.max(c_1[:, 0]), np.max(c_2[:, 0])),
```

```
                    step=1)

line_y = - (w[0] * line_x) / w[1]
plt.plot(line_x, line_y)
plt.show()
```

5.7.2 线性判别分析算法评价

（一）优点

- 利用类别信息：LDA 是一种有监督的降维技术，它充分利用了数据的类别标签信息，在降维的同时能够有效区分不同类别的数据，这对于分类任务非常有帮助，能够提高分类模型的性能。
- 可解释性强：投影方向是通过对类内和类间散度矩阵的分析得到的，具有明确的物理意义，可解释性强。可以通过分析投影方向来理解不同特征对分类的贡献程度。

（二）缺点

- 计算复杂度高：计算类内散度矩阵和类间散度矩阵需要遍历所有数据点，并且求解广义特征值问题的计算量也较大，因此 LDA 的计算复杂度较高，在处理大规模数据集时效率较低。
- 易过拟合：当样本数量较少或者类别之间的方差差异很大时，LDA 容易出现过拟合现象。因为在这种情况下，估计的类内和类间散度矩阵可能不准确，导致投影方向不能很好地反映数据的真实分布。

5.7.3 实践案例：基于线性判别分析的三维数据集降维

（一）数据准备与分析

在 Python 中，通过调用 Scikit - learn 库中 datasets 模块的 make_classification 函数构建三维空间中的数据集。该数据集共有 1000 个数据点，每个数据点都有一个类别标签，共有 3 个类别。然后使用 Scikit - learn 中 discriminant_analysis 模块的 LinearDiscriminantAnalysis 函数实现 LDA 算法。首先创建 LinearDiscriminantAnalysis 对象并设置参数，然后调用 fit_transform 方法拟合数据并进行降维。代码如下：

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.datasets import make_classification
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
#1.生成三维数据集
X, y = make_classification(n_samples=1000, n_features=3, n_informative=2, n_redundant=0, n_repe
ated=0,
```

```
                                n_classes=3, n_clusters_per_class=1, class_sep=0.5, random_state=10)


plt.rcParams['font.sans-serif']=['Times New Roman']
plt.rcParams.update({'font.size': 35})
fig = plt.figure(figsize=(20, 10))
#2.绘制原始数据集的散点图
ax = fig.add_subplot(121, projection='3d')
shapes = ['o', '^', 's']
colors = ['blue', 'green', 'yellow']
color_mapping = {0: colors[0], 1: colors[1], 2: colors[2]}
for i in range(3):
    markers = shapes[i]
    scatter_colors = color_mapping[i]
    ax.scatter(X[y == i, 0], X[y == i, 1], X[y == i, 2], marker=markers, c=scatter_colors, label=f'Class {i}')
ax.legend(frameon=False)
#3.调用LDA算法进行降维，并拟合数据
lda = LinearDiscriminantAnalysis(n_components=2)
X_new = lda.fit_transform(X, y)
plt.subplot(122)
for i in range(3):
    markers = shapes[i]
    scatter_colors = colors[i]
    plt.scatter(X_new[y == i, 0], X_new[y == i, 1], marker=markers, c=scatter_colors, label=f'Class {i}')
plt.legend(frameon=False)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.tight_layout()
plt.show()
```

（二）结果分析

通过上述代码，对三维数据集进行降维处理。降维后的结果可以用于可视化，观察不同类别的数据在二维空间中的分布情况。在降维后的二维空间中，如果不同类别的数据点能够明显分开，说明 LDA 算法能够有效地找到一个投影方向，使得不同类别的数据点在投影后具有很好的可分性。同时，也可以将降维后的数据用于后续的分类任务，如使用逻辑回归等分类算法进行训练和预测，评估 LDA 降维对分类性能的影响。

5.8 t-SNE

5.8.1 t-SNE基本原理

（一）基本思想

t-SNE算法是在SNE算法基础上改进而来，SNE算法的思想主要包含如下两步：

- 利用高维空间中的点组成的组合，在此基础上构造一个概率分布：如果组合具有比较大的相似性，就赋予一个较大的概率，反之则赋予一个较小的概率；
- 基于低维空间中的点组成的map构造一个相似的概率分布，然后在这个map的点的取值范围内对这两个分布之间的KL散度进行优化使之达到极小。

t-SNE（t-Stochastic Neighbor Embedding）是一种强大的非线性降维算法，专注于数据的局部结构，旨在通过最小化高维空间和低维空间中数据点相似度的差异来实现数据的非线性降维。它的核心在于利用样本数据邻近点的分布进行降维操作，通过将欧氏距离转换为条件概率分布，用条件概率来描述数据点之间的相似性。
给定一组高维空间中的数据矩阵，t-SNE 以每个数据点为中心构建高斯分布，根据其他数据点在该高斯分布中的概率来确定其作为邻近点的条件概率，从而构建高维空间数据点的条件概率分布 P。在低维空间中，同样使用条件概率描述数据点之间的相似性，构建条件概率分布 Q。t-SNE 的目标是最小化高维空间和低维空间中概率分布的 KL 散度，使得降维后的数据能最大程度保持原数据点之间的相似性，尤其是局部相似性。其中，KL散度（Kullback-Leibler divergence），可以称作相对熵（relative entropy）或信息散度（information divergence）。KL散度的理论意义在于度量两个概率分布之间的差异程度，当KL散度越大的时候，说明两者的差异程度越大；而当KL散度小的时候，则说明两者的差异程度小。如果两者相同的话，则该KL散度应该为0。
（二）对SNE算法的改进
针对SNE算法的两个困难：

1. 损失函数难以优化；
2. crowding问题；

t-SNE算法进行了两项改进：

1. 使用一个对称性的损失函数，因为KL散度是不对称的；
2. 利用t分布而不是高斯分布来衡量点在低维空间中的相似性；

和高斯分布相比,t分布有一个较长的尾巴，这使得t-SNE算法能够有效地改善优化过程与crowding问题。简单解释一下crowding问题：高维空间中的相邻关系在低维空间中无法完全保留。例如：在二维空间中，等边三角形的三个顶点，它们彼此的距离相等，但是在一维空间中不存在这样的三个点。再如，在高维空间中与xi具有相似位置关系的点太多，所以变换到低维空间后，以yi为中心的领域根本无法容纳如此多的点，以致于在高维空间中相

距中等的点在低维空间中的距离要变得非常大。

（三）实现步骤

- 对于给定的高维数据集，根据公式构建高维空间的条件概率分布。
- 在低维空间中，根据 t 分布函数构建条件概率分布。
- 计算联合概率，并构建目标函数。
- 使用梯度下降法不断调整低维数据点的位置，最小化目标函数，从而优化数据点在低维空间中的分布，以更好地反映高维空间中的局部结构。

下面对scikit-learn包含的一个手写数字数据集应用t-SNE流形学习算法。在这个数据集中，每个数据点都是0到9之间手写数字的一张 8×8 灰度图像。下面观察给出的每个类别：

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits

digits = load_digits()
fig, axes = plt.subplots(2, 5, figsize=(10, 5), subplot_kw={'xticks':(), 'yticks': ()})
for ax, img in zip(axes.ravel(), digits.images):
    ax.imshow(img)
plt.show()
```

我们用PCA将降到二维的数据可视化。对前两个主成分作图，并按类别对数据点着色：

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA

digits = load_digits()
# 构建一个PCA模型
pca = PCA(n_components=2)
pca.fit(digits.data)
# 将digits数据变换到前两个主成分的方向上
digits_pca = pca.transform(digits.data)
colors = ["#476A2A", "#7851B8", "#BD3430", "#4A2D4E", "#875525", "#A83683", "#4E655E", "#85354
1", "#3A3120", "#535D8E"]
plt.figure(figsize=(10, 10))
plt.xlim(digits_pca[:, 0].min(), digits_pca[:, 0].max())
```

```
plt.ylim(digits_pca[:, 1].min(), digits_pca[:, 1].max())
for i in range(len(digits.data)):
    # 将数据实际绘制成文本，而不是散点
    plt.text(digits_pca[i, 0], digits_pca[i, 1], str(digits.target[i]),
    color = colors[digits.target[i]],
    fontdict={'weight': 'bold', 'size': 9})
plt.xlabel("First principal component")
plt.ylabel("Second principal component")
plt.show()
```

实际上，这里我们用每个类别对应的数字作为符号来显示每个类别的位置。利用前两个主成分可以将数字 0、6 和4相对较好地分开，尽管仍有重叠。大部分其他数字都大量重叠在一起。我们将 t-SNE应用于同一个数据集，并对结果进行比较。由于t-SNE不支持变换新数据，所以TSNE类没有 transform方法。我们可以调用 fit_transform方法来代替，它会构建模型并立刻返回变换后的数据：

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.manifold import TSNE

digits = load_digits()
colors = ["#476A2A", "#7851B8", "#BD3430", "#4A2D4E", "#875525", "#A83683", "#4E655E", "#85354
1", "#3A3120", "#535D8E"]

tsne = TSNE(random_state=42)
# 使用fit_transform而不是fit，因为TSNE没有transform方法
digits_tsne = tsne.fit_transform(digits.data)
plt.figure(figsize=(10, 10))
plt.xlim(digits_tsne[:, 0].min(), digits_tsne[:, 0].max() + 1)
plt.ylim(digits_tsne[:, 1].min(), digits_tsne[:, 1].max() + 1)
for i in range(len(digits.data)):
    # 将数据实际绘制成文本，而不是散点
    plt.text(digits_tsne[i, 0], digits_tsne[i, 1], str(digits.target[i]),
    color = colors[digits.target[i]],
    fontdict={'weight': 'bold', 'size': 9})
plt.xlabel("t-SNE feature 0")
```

```
plt.xlabel("t-SNE feature 1")
plt.show()
```

t-SNE的结果非常棒。所有类别都被明确分开。数字1和9被分成几块，但大多数类别都形成一个密集的组。要记住，这种方法并不知道类别标签：它完全是无监督的。但它能够找到数据的一种二维表示，仅根据原始空间中数据点之间的靠近程度就能够将各个类别明确分开。t-SNE算法有一些调节参数，虽然默认参数的效果通常就很好。可以尝试修改perplexity和early_exaggeration，但作用一般很小。

5.8.2 t-SNE算法评价

（一）优点

- 强大的非线性降维能力：t-SNE 能够有效处理复杂的非线性数据，在保持数据局部结构方面表现出色。它可以将高维数据中紧密相连的局部区域在低维空间中也映射为相近的区域，非常适合用于探索数据的局部特征和聚类结构。
- 良好的可视化效果：常用于数据可视化和探索性数据分析（EDA）。在二维或三维空间中展示数据点之间的相对关系时，t-SNE 能够将具有相似特征的数据点聚集在一起，不同类别的数据簇相距较远，使得数据的分布结构一目了然，有助于发现数据中的隐藏模式和类别分布。

（二）缺点

- 计算复杂度高：t-SNE 算法在计算条件概率分布和进行梯度下降优化时，计算量较大，尤其是在处理大规模数据集时，计算时间会显著增加，对计算资源的要求较高。
- 对异常值敏感：由于 t-SNE 依赖数据点之间的距离来构建概率分布，异常值的存在会对距离计算产生较大影响，进而干扰条件概率分布的构建，导致降维结果可能受到异常值的强烈干扰，影响对数据真实结构的判断。
- 无法保留全局数据结构：t-SNE 主要关注数据的局部结构，在降维过程中通常无法保留全局数据结构。在一些需要考虑数据整体分布和关系的场景下，t-SNE 的表现可能不佳。

5.8.3 实践案例：基于t-SNE的手写数字数据集降维

（一）数据准备和分析

在 Python 中，通过调用 Scikit-learn 库中 datasets 模块的 load_digits 函数加载手写数字图片数据集。该数据集常用于机器学习和模式识别领域，包含从 0 到 9 共十类数字，每个样本是一个图像，被展平为一个长度为 64 的特征向量（8x8 = 64），即图像的每个像素点信息均为高维空间中的一个特征。本案例中选取从 0 到 4 共五类数字。之后使用 Scikit-learn 中 manifold 模块的 TSNE 函数实现 t-SNE 算法。首先创建对象并设置参数，然后调用 fit_transform 方法拟合数据并进行降维。示例代码如下：

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import manifold, datasets
#1.构建数据集
digits = datasets.load_digits(n_class=5)
X, y = digits.data, digits.target
n_samples, n_features = X.shape
n = 15
img = np.zeros((10 * n, 10 * n))
#2.可视化原始数据集
for i in range(n):
    ix = 10 * i + 1
    for j in range(n):
        iy = 10 * j + 1
        img[ix:ix + 8 , iy:iy +8 ] = X[i * n + j].reshape((8,8))
plt.figure(figsize=(8,8))
plt.imshow(img, cmap=plt.cm.binary)
plt.xticks([])
plt.yticks([])
plt.show()
# 3.调用TNE函数，拟合数据
tsne = manifold.TSNE(n_components=2, random_state=501)
X_tsne = tsne.fit_transform(X)
x_min, x_max = X_tsne.min(0), X_tsne.max(0)
X_norm = (X_tsne - x_min) / (x_max - x_min)
#4.可视化降维结果
plt.figure(figsize=(8,8))
for i in range(X_norm.shape[0]):
    plt.text(X_norm[i,0], X_norm[i,1], str(y[i]), color=plt.cm.Set1(y[i]), fontdict={'weight':
'bold', 'size': 9})
plt.xticks([])
plt.yticks([])
plt.show()
```

（二）结果分析

通过上述代码，对手写数字数据集进行降维处理。降维后的结果可以使用 matplotlib 等绘图库进行可视化。在二

维可视化图中，可以观察到不同类别的数字数据点在空间中的分布情况。通常，具有相似特征的数字会聚集在一起形成不同的簇，不同类别的簇之间会有一定的间隔，这表明 t-SNE 能够有效地区分不同类别的数据点，提高数据的可分离性。通过调整 t-SNE 的参数，如困惑度、学习率等，可以观察到降维结果的变化，从而选择最适合数据集的参数设置，以获得更好的可视化效果和对数据结构的理解。

5.9 UMAP

5.9.1 UMAP算法基本思想

（一）基本思想

UMAP（Uniform Manifold Approximation and Projection）是一种在降低数据样本维数的同时，尽可能最大限度地保留数据的全局结构特性的降维技术。它与 t-SNE 算法类似，先构建高维空间的局部结构，但在保留数据结构方面有独特的方法。UMAP 基于流形学习的概念，认为高维数据实际上是由一个低维流形映射而来，虽然数据在高维空间分布复杂，但局部仍具有欧氏空间的性质。

UMAP 通过构建高维空间和低维空间的条件概率分布函数，构造目标函数来度量两个概率分布之间的相似性，然后使用梯度下降等优化方法最小化目标函数，从而获得最优的低维表示，实现降维的同时保留数据的局部和全局结构。

这些定义比较官方，以下描述是总结出来的可帮助我们理解 UMAP 的要点。

- Projection ——通过投影点在平面、曲面或线上再现空间对象的过程或技术。也可以将其视为对象从高维空间到低维空间的映射。
- Approximation——算法假设我们只有一组有限的数据样本（点），而不是构成流形的整个集合。因此，我们需要根据可用数据来近似流形。
- Manifold——流形是一个拓扑空间，在每个点附近局部类似于欧几里得空间。一维流形包括线和圆，但不包括类似数字8的形状。二维流形（又名曲面）包括平面、球体、环面等。
- Uniform——均匀性假设告诉我们我们的数据样本均匀（均匀）分布在流形上。但是，在现实世界中，这种情况很少发生。因此这个假设引出了在流形上距离是变化的概念。即，空间本身是扭曲的：空间根据数据显得更稀疏或更密集的位置进行拉伸或收缩。

综上所述，我们可以将UMAP描述为:

一种降维技术，假设可用数据样本均匀（Uniform）分布在拓扑空间（Manifold）中，可以从这些有限数据样本中近似（Approximation）并映射（Projection）到低维空间。上面对算法的描述可能会对我们理解它的原理有一点帮助，但是对于UMAP是如何实现的仍然没有说清楚。为了回答“如何”的问题，让我们分析UMAP执行的各个步骤。

（二）实现步骤

- 根据高维数据集确定近邻点个数，计算每个数据点的最近邻值以及高维空间的条件概率分布函数。
- 设置低维空间条件概率分布函数的参数和，构建低维空间的条件概率分布函数。

- 构建目标函数。
- 使用梯度下降等优化方法最小化目标函数，得到数据在低维空间的最优表示。

（三）思考比较t-SNE和UMAP的异同

t-SNE（t - 分布随机邻域嵌入）和 UMAP（均匀流形近似和投影）都是在数据降维与可视化领域广泛应用的算法，二者在功能上有相似之处，但原理和特性上存在诸多差异。

1. 相同点

- 目的：二者均用于数据降维，旨在将高维数据映射到低维空间，以便数据可视化和探索性分析，帮助用户理解数据的内在结构、分布和关系。在处理高维的图像、文本数据时，可将其降维至二维或三维空间进行可视化展示 。
- 适用场景：都适用于处理非线性数据，在数据具有复杂的非线性结构时，能有效挖掘数据的局部特征和结构，将相似的数据点在低维空间中映射得更接近。
- 基于概率分布：都基于概率分布来衡量数据点之间的相似性。t-SNE 通过构建高维与低维空间的条件概率分布，用条件概率描述数据点相似性；UMAP 通过构建高维空间的局部结构和低维空间的条件概率分布函数，度量数据点相似性。

2. 不同点

- 原理差异：t-SNE 基于随机邻域嵌入，通过最小化高维与低维空间中数据点相似度的差异（以 KL 散度度量）实现降维，在高维空间用高斯分布构建条件概率，低维空间用 t 分布构建，解决数据 “拥挤” 问题。UMAP 基于流形学习，构建高维空间局部流形，通过梯度下降优化目标函数保留数据全局结构特性，引入超参数调节低维空间距离对应关系。
- 计算复杂度：t-SNE 计算复杂度较高，计算条件概率分布和梯度下降优化时计算量大，处理大规模数据耗时久。UMAP 相对高效，简化计算过程，直接使用联合概率减少计算量和复杂度，处理大规模数据优势明显。
- 对数据结构的保留：t-SNE 主要保留局部结构，能将高维数据的局部相似性较好地映射到低维空间，但全局结构保留不佳。UMAP 兼顾局部和全局结构，通过目标函数中的两项分别保持局部和全局结构信息，在低维空间中能更全面反映数据的结构关系。
- 超参数敏感性：t-SNE 超参数主要有困惑度、学习率等，对结果有影响，但敏感性相对弱些。UMAP 对邻域大小（值）、最小距离和其它超参数选择非常敏感，不同超参数设置可能导致差异较大的降维结果，需多次实验调优。
- 应用场景侧重：t-SNE 常用于数据可视化和探索性数据分析，在研究数据聚类结构和类别分布方面应用广泛。UMAP 在聚类分析、可视化和特征提取方面均表现出色，因其高效性和全局结构保留能力，在处理大规模数据和需要全面把握数据结构的场景中更具优势。

5.9.2 UMAP算法评价

（一）优点

1. 有效处理非线性数据：UMAP 特别适用于高维数据的降维，尤其是当数据具有非线性结构和复杂的局部关系时。它能够通过构建局部流形和合适的概率分布，有效地保留数据的局部和全局结构，在聚类分析、可视化和特征提取方面表现出色。
2. 计算效率高：相比一些其他非线性降维算法，UMAP 在计算上更为高效。它简化了计算过程，特别是在处理大规模数据时，优势更加明显。通过直接使用联合概率，减少了计算量和复杂度，提高了算法的运行速度。
3. 适用性广泛：适用于多种领域的数据降维任务，如生物学、图像处理、文本分析等。在探索性数据分析和模式识别任务中，能够帮助发现数据中的潜在模式和结构，为后续的数据分析和决策提供有力支持。

（二）缺点

1. 对超参数敏感：UMAP 算法对于邻域大小、最小距离和其它超参数的选择非常敏感。不同的超参数设置可能会导致差异较大的降维结果，需要进行多次实验和调优才能找到合适的参数组合，这在一定程度上增加了使用的难度。
2. 结果解释性有限：虽然 UMAP 能够有效地降维和保留数据结构，但降维后的结果在解释性方面存在一定的局限性。与一些基于线性变换的降维方法相比，很难直观地解释低维空间中的维度与原始数据特征之间的关系。

5.9.3 实践案例：基于UMAP的手写数字数据集降维

（一）数据准备和分析

本案例使用 UMAP 算法对 5.8.3 案例中的手写数据集进行降维。该手写数字数据集包含从 0 到 9 的手写数字图像，每个图像被展平为一个特征向量。在 Python 中，通过调用 Scikit-learn 库中 datasets 模块的 load_digits 函数加载数据集，并进行相应的预处理。之后在 Python 中，UMAP 算法可以通过调用 Scikit-learn 中 umap 模块的 UMAP 函数实现。首先创建对象并设置参数，然后调用 fit_transform 方法拟合数据并进行降维。示例代码如下：

```
from umap import UMAP
from sklearn.manifold import TSNE
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import load_digits
import numpy as np
import matplotlib.pyplot as plt
# 1. 生成数据集
digits = load_digits(n_class=8)
X, y = digits.data, digits.target
```

```
n_samples, n_features = X.shape
n = 5000
X, y = X[:n], y[:n]
# 2. 调用UMAP算法将数据集降至二维
reducer_umap = UMAP(n_components=2, random_state=0)
embedding_umap = reducer_umap.fit_transform(X)
scaler_umap = MinMaxScaler()
embedding_umap = scaler_umap.fit_transform(embedding_umap)
# 3. 调用t-SNE算法将数据集降至二维
reducer_tsne = TSNE(n_components=2, random_state=0)
embedding_tsne = reducer_tsne.fit_transform(X)
scaler_tsne = MinMaxScaler()
embedding_tsne = scaler_tsne.fit_transform(embedding_tsne)
# 4. 可视化UMAP算法结果
plt.rcParams['font.sans-serif']=['Times New Roman']
plt.rcParams.update({'font.size': 35})
fig = plt.figure(figsize=(20, 10))
plt.subplot(1, 2, 1)
plt.title('UMAP')
for i in range(embedding_umap.shape[0]):
 plt.text(
 embedding_umap[i, 0],
 embedding_umap[i, 1],
 str(y[i]),
 color=plt.cm.tab10(y[i]),
 fontdict={"weight": "bold"},
 va="center",
 ha="center",
)
plt.scatter(embedding_umap[:, 0], embedding_umap[:, 1], c=y, cmap='viridis', s=30, edgecolors
='k')
for digit in range(len(np.unique(y))):
 digit_indices = np.where(y == digit)
 plt.text(
 np.mean(embedding_umap[digit_indices, 0]),
 np.mean(embedding_umap[digit_indices, 1]),
 str(digit),
```

```
 color='black',
 va="center",
 ha="center",
)
plt.tick_params(axis='both', which='major')
# 5. 可视化t-SNE算法结果
plt.subplot(1, 2, 2)
plt.title('t-SNE')
for i in range(embedding_tsne.shape[0]):
 plt.text(
 embedding_tsne[i, 0],
 embedding_tsne[i, 1],
 str(y[i]),
 color=plt.cm.tab10(y[i]),
 va="center",
 ha="center",
)
plt.scatter(embedding_tsne[:, 0], embedding_tsne[:, 1], c=y, cmap='viridis', s=30, edgecolors
='k')
for digit in range(len(np.unique(y))):
 digit_indices = np.where(y == digit)
 plt.text(
 np.mean(embedding_tsne[digit_indices, 0]),
 np.mean(embedding_tsne[digit_indices, 1]),
 str(digit),
 color='black',
 va="center",
 ha="center",
)
plt.tick_params(axis='both', which='major')
plt.tight_layout()
plt.show()
```

（二）结果分析

通过上述代码，对手写数字数据集进行降维处理。降维后的结果可以使用 matplotlib 等绘图库进行可视化。在二维可视化图中，可以观察到不同类别的数字在低维空间中的分布情况。通常，UMAP 算法能够将不同类别的数字进行分离，形成清晰的簇，使其更容易被区分。与 t-SNE 算法相比，UMAP 算法形成的簇间距离更大，簇内距

离更小，展示出了更好的分类效果。但由于 UMAP 算法对超参数敏感，不同的参数设置可能会导致降维结果的差异，因此可以通过调整参数来观察算法的分类结果，进一步优化降维效果。